



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Centre de Formació Interdisciplinària Superior



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona



telecos
BCN



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona



UNIVERSITY OF
TORONTO

Bachelor's degree thesis

Joint object boundary and skeleton detection using convolutional neural networks

Carles Balsells Rodas

Supervised by:

Sven Dickinson (UofT)

Stavros Tsogkas (UofT)

Lluís A. Belanche (UPC)

In partial fulfillment of the requirements for the

Bachelor's degree in Informatics Engineering

Bachelor's degree in Engineering Physics

July 2019

Abstract

Joint object boundary and skeleton detection using convolutional neural networks

by Carles Balsells Rodas

We address object boundary and skeleton detection. Contrary to recent approaches, which tackle both tasks separately, we aim to address them simultaneously. Our motivation is based on observing that object boundaries and skeletons are dual representations of the shape and symmetries of an object. A deep-learning-based method is presented, our joint medial axis and boundary detector. It uses a shared feature representation to extract both object boundaries and skeletons from a natural scene. We also propose and test an extension to enforce consistency between both tasks. We test our approach using COCO, a challenging dataset which provides natural scenes with different complexity. Given our scenario, we see that our joint medial axis and boundary detector is able to improve performance over detecting boundaries and skeletons separately. However, the consistency extension is not able to boost performance on the previous model.

Keywords: computer vision, convolutional neural network, edge, boundary, skeleton, medial axis, side outputs, fully convolutional network

Acknowledgements

I would like to express my sincere gratitude to both prof. Sven Dickinson and Stavros Tsogkas for giving me the opportunity of being an active collaborator of their field of research and also being part of their team during the time of my stay. They have been interested in me growing both in an academic and personal level. I would also like to thank the colleagues of my work group, who have helped me facing the difficulties that I have encountered while moving on with this project and have also been friendly sharing some free-time of theirs with me.

Moreover, I would like to thank Lluís A. Belanche to accept carrying out the supervision of my work directly from UPC.

This visiting experience could not have taken place without the support from CFIS, who established the first contact to my supervisors in University of Toronto last year. Special thanks also to Fundació Privada Cellex and Generalitat de Catalunya for supporting my stay financially.

Preface

This report represents the work done as a visiting student at University of Toronto from February to July 2019. During the stay, prof. Sven Dickinson and Stavros Tsogkas have been in charge of supervising the project. Similarly, Lluís A. Belanche from UPC has been the contact to my home university and has also been in charge of supervising this work.

During this 5-month stay, I have been able to learn about applying modern convolutional-neural-network-based approaches to the field of computer vision. This experience as a visiting student has been enriching in all aspects. Not only have I gained experience in an academic level as a researcher but also in a more personal one due to the fact of going abroad. I am proud of presenting this report as the Bachelor's Thesis of my undergraduate studies as I have been able to explore edge and skeleton detection tasks bringing new ideas that have not been reported in the past.

Contents

List of Figures	vii
1 Introduction	1
1.1 Description and motivation	1
1.2 How will we address these tasks?	2
1.3 Outline	3
2 Previous work	4
2.1 Edge detection	4
2.2 Skeleton detection	5
3 Overview of Convolutional Neural Networks	7
3.1 Layers used in our implementation	9
3.1.1 Convolution	10
3.1.2 Transposed convolution	12
3.1.3 Non-linear activations	13
3.1.4 Residual	13
3.1.5 Pooling	14
3.2 Parameter optimization	15
4 Joint learning of object boundaries and skeletons	17
4.1 Holistically-nested Edge Detection	17
4.1.1 Formulation	18
4.1.2 Implementation	21
4.2 Deep Skeleton	23
4.2.1 Formulation	24
4.2.2 Implementation	30

4.3	Joint edge and skeleton detector	32
4.3.1	Formulation	32
4.3.2	Implementation	33
4.4	Enforcing consistency between both tasks	34
4.4.1	Formulation	35
4.4.2	Conditional Adversarial Network	37
5	Experiments	43
5.1	Datasets	43
5.1.1	BSDS500	43
5.1.2	SK-LARGE	44
5.1.3	COCO	45
5.2	Performance evaluation protocol	46
5.3	Re-implementations	48
5.3.1	HED	48
5.3.2	LMSDS	51
5.4	Joint edge and skeleton extraction	55
5.4.1	Training scheme	57
5.4.2	Results	60
6	Conclusions and future work	77
	Bibliography	77

List of Figures

1.1	Example showing some segmented shapes. The medial axis points of the boundaries of these shapes creates its skeleton.	2
3.1	Example of a simple neural network consisting of one input layer of 3 neurons, several (n) hidden layers of 4 neurons and an output layer of 2 neurons. . . .	8
3.2	Example of a network operation in a fully connected layer. x is the input, w is the weight, b is the bias added to the sum and f is the activation function. . . .	8
3.3	Visual example of the 3D arrangement of a CNN.	9
3.4	Example of a 3x3 convolution with stride 1 and no padding applied. As we can see, the kernel slides along the input feature map, performs a dot product and generates an output feature map as a result.	11
3.5	Illustration of the comparison between a convolution and a transposed convolution. The latter aims to learn an upsampling kernel to reconstruct the input.	12
3.6	Representation of a residual block, where convolutional layers and non-linearities (ReLU) are combined.	14
3.7	Example of a maxpool layer consisting of a 2x2 filter with stride 2.	15
3.8	Example to understand the intuition of backpropagation. The forward computation is indicated in green, starting at x , y , z and t with values 3, -4 , 2 and -1 respectively. The backward one is indicated in red, starting with 1 at the last step. By performing a recursive derivative chain rule in backwards direction we can compute the gradients of the four input values.	16
4.1	Example of the HED algorithm response. (a) shows an example image from BSDS500 [4]. (b) shows its corresponding edges annotated by humans. (c) corresponds to the HED output. (d), (e), and (f) show the side-output responses at different stages.	18

4.2	Holistically-nested network architecture.	19
4.3	Illustration of the HED architecture implementation. One upsampling layer is added after each side output layer to match the size of the input image. The fusion layer learns how to combine responses from multiple scales automatically. The dashed lines at each side output indicate supervision performed by $\ell_{side}^{(1)}$, $\ell_{side}^{(2)}$, $\ell_{side}^{(3)}$, $\ell_{side}^{(4)}$ and $\ell_{side}^{(5)}$ respectively. The one at the fusion layer level corresponds to \mathcal{L}_{fuse}	22
4.4	Example of the skeleton extraction at different parts of an object for different filter sizes. If we are able to capture the boundaries of the object, we can extract its medial axis points. In our network, the receptive field size increases at each stage, and therefore, the range for skeleton extraction. . . .	24
4.5	Example of the generation of the scale map Z . The input image X has a corresponding ground truth skeleton map Y (first image in blue) and scale map S (middle image). The values in which the quantized scale map Z can be labeled are represented different colors, which refer to different ranges of scales. . . .	25
4.6	Example of the computation of the scale-associated side outputs. Here, stage 3 is shown. Thus, $i = 3$. a_{jk}^i represent the activations of the skeleton localization side output. The ground truth skeleton map represents the quantized scale map at this stage $Z^{(i)}$	26
4.7	Illustration of the fusion of a LMSDS network with 4 stages.	28
4.8	Illustration of the LMSDS architecture implementation. As we mentioned, VGG16 is the backbone network of the skeleton detector, which results in 4 different stages. Each stage adds a quantized scale label and we perform supervision for both skeleton localization and scale regression with respect to the quantized ground truth scale map. We must not forget about the fusion layer (figure 4.7) and its corresponding supervision (not represented here). . . .	30
4.9	Illustrative example of the skeleton localization responses at each stage. The fusion output is also represented and arranges the side output classification to provide a global result.	31
4.10	Illustration of the implementation of our joint medial axis and boundary detector. We do not include supervision as we can extrapolate it from the previous architectures (figures 4.3 and 4.8). We must not forget about the fusion layers for the edge and skeleton detection.	34

4.11	Illustration of the consistency module extension performed by image-to-image translations.	36
4.12	Illustration of the idea behind the consistency loss. Our intention is that the edge and skeleton predictions should be consistent when translating them into each other's domain.	36
4.13	Illustration of the generator architecture, composed by three stages: encoding (convolutions), domain translation (9 residual blocks) and decoding (deconvolutions). This setting is identical for both domain translations.	39
4.14	Illustration of the discriminator architecture, also called PatchGAN. It performs the discrimination on squared regions with the size of the receptive field size of the network. The red box shows that we add one extra layer for the edge to skeleton mapping because it gives better translations.	40
4.15	Some examples of the domain translations. Each row represents a paired edge and skeleton map. The first column shows the original boundary map; the second one is the result of mapping its respective skeleton map to the edge domain; the third one represents the original skeleton map; and the last column is the generated skeleton from its respective ground truth boundary map. . .	42
5.1	Sample from BSDS500 dataset which shows a natural image with five different human annotations for edges, denoted in white.	44
5.2	Some examples of SK-LARGE dataset. The skeleton ground truth is indicated in red.	45
5.3	Several samples from COCO dataset where several people appear. These people "instances" have been segmented from the original image.	46
5.4	Example of the edge detection performance evaluation protocol for a single image of the test set. The valid F-score is the one which is best for a certain threshold applied to the thinned response map.	47
5.5	Example of the brute-force force approach taken to generate the ground truth set. The image on the left shows all the human annotations put together. . .	49
5.6	Comparison of our results on the BSDS500 dataset with respect to the original ones.	50
5.7	Some examples of the edge map responses obtained by our version of the HED after applying nms.	50

5.8	Example of the value of the objective function with respect to the epochs run in the training phase.	53
5.9	Precision-recall curve comparison between our LMSDS re-implementation and the original version, both tested on SK-LARGE dataset.	56
5.10	Some examples of skeleton extractions performed by our version of LMSDS after applying nms.	56
5.11	Illustration of the ground truth generation procedure referred to COCO dataset. Given a natural scene, we take its annotations from COCO to obtain object segmentations which are used to generate boundaries and skeletons in a deterministic fashion.	58
5.12	Results of the edge detection task tested on the entire COCO test set that has been defined.	61
5.13	Results of the skeleton detection task tested on the entire COCO test set that has been defined.	61
5.14	Precision-recall curves of the models defined for the edge detection task. The results correspond to the first scenario.	62
5.15	Precision-recall curves of the models defined for the skeleton detection task. The results correspond to the first scenario.	62
5.16	Samples of the edge detectors in the 1-segmentation scenario. We can see that when the scene shows more elements, the detectors fire incorrect edges. Moreover, the detectors are able to capture the boundary of the object without firing internal edges. Notice also that the responses of the consistency module look thicker than the first ones.	63
5.17	Samples of the skeleton detectors in the 1-segmentation scenario. All of them are able to extract object skeletons. Notice that in general, our joint detector provides more precision than the other ones.	64
5.18	Results of the edge detection task evaluated on natural scenes that contain between 2 and 5 segmentations.	65
5.19	Results of the skeleton detection task evaluated on natural scenes that contain between 2 and 5 segmentations.	65

5.20	Samples of the edge detectors in the 2-to-5-segmentation scenario. This scenes look more complex than the last ones but the detectors are capable of extracting boundaries from the people that appear in the image. We can also see that our joint network with the consistency module becomes very innacurate when the detections to fire are very small.	66
5.21	Samples of the skeleton detectors in the 2-to-5-segmentation scenario. Even though the task is getting more difficult, they are able to fire the main medial points. We can see that the skeletons that both joint approaches provide are more complete than the LMSDS (second row).	67
5.22	Results of the edge detection task evaluated on natural scenes that contain more than 5 segmentations.	68
5.23	Results of the skeleton detection task evaluated on natural scenes that contain more than 5 segmentations.	68
5.24	Samples of the skeleton detectors in the more-than-5-segmentation scenario. We can see that the dectectors fire all the labeled objects that appear in the scene. However, our joint network with the consistency extension provides less accuracy, which results in HED outperforming it.	69
5.25	Samples of the skeleton detectors in the more-than-5-segmentation scenario. Despite the complexity that these natural scenes show, the skeleton detectors are able to fire the main symmetry points of the labels objects. In general, we see that our joint detector is more accurate.	70
5.26	Results of the edge detection task evaluated on natural scenes where we can find segmentations wider than 196 pixels.	71
5.27	Results of the skeleton detection task evaluated on natural scenes that produce ground truth scale maps with scales larger than 196 pixels.	71
5.28	Samples of the edge detectors in natural scenes with scales higher than 196 pixels. Both joint approaches outperform HED. We can see the latter fires more internal edges. Although they are not supposed to provide decent detections, we can see that they look decent enough.	72
5.29	Samples of the skeleton detectors in natural scenes with scales higher than 196 pixels. We can clearly see that they are able to fire some medial axis points of the largest segmentation but in a very inaccurate manner, which results in this performance drop observed in the precision-recall curve.	74

5.30 Examples showing the consistency module (one for each row). As we can see, the mapping functions let us obtain edge maps that correspond to the original ones only with information about skeleton and scales. The skeleton maps obtained from object boundaries look less accurate but look decent enough to enforce consistency. Notice that are still able to obtain decent translations when dealing with crowded scenes (last row). 75

5.31 Examples of segmentation reconstruction for each object that appears in the scene. As we can see, all the models are able to segment all the objects. In general, our joint approach with the consistency module is able to provide more accurate results. 76

Chapter 1

Introduction

1.1 Description and motivation

In this report we face two main computer vision problems addressed by a great number of researchers in the past: edge detection and skeleton detection.

Edges are classically defined as intensity changes, which occur in a natural image over a wide range of scales. These intensity changes arise from surface discontinuities or from reflectance or illumination boundaries, and these all have the property that they are spatially localized [30]. The goal of edge detection is to find these abrupt changes in a natural image. The wide range of scales in which an edge can appear goes from a local level, such as internal edges in an object, from a more global one, such as object boundaries. We are interested in this last type of edges as they provide information about object shape, symmetry, etc.

The skeleton, also called symmetry axis, is a structure that captures geometric and topological properties of shapes and object boundaries [13]. In spite of its inherent instability, it has found applications in a number of areas that deal with shapes. Therefore, skeleton detection is of great importance so as to encode an object based on its symmetries. However, detecting medial axis points on a natural image is a very challenging task. In figure 1.1, we have sketched some boundaries together with their respective medial axes so as to illustrate the previous description.

Object skeletons and object boundaries are dual representations of shape. If we know the object boundaries, we can build its object skeleton by performing a medial axis transform [13]. On the other hand, we can reconstruct the shape of an object given its skeleton, although

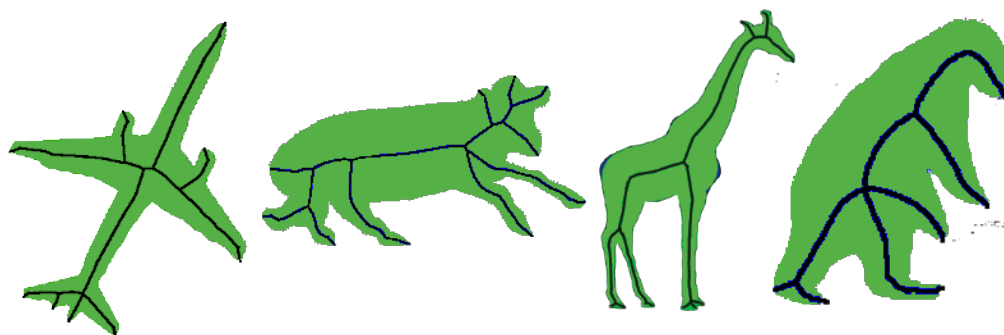


Figure 1.1: Example showing some segmented shapes. The medial axis points of the boundaries of these shapes creates its skeleton.

there still lacks information about the scale of the symmetry axis, i.e. the distance from a medial axis point to the closest boundary. Our work is motivated by exploring this idea of the duality that is shared between object boundaries and skeletons.

As a result, our hypotheses is the following: if we address both tasks using a deep-learning-based approach, a shared feature representation should improve performance in both cases. In other words, we aim to develop a single convolutional neural network capable of extracting simultaneously boundaries and skeletons.

1.2 How will we address these tasks?

First of all, we will present an edge detection algorithm called Holistically-nested Edge Detection (HED). HED is used to extract edges from a natural image so as to obtain an edge extraction map. We will be using this algorithm to extract boundaries from objects in a natural scene.

Later, we will talk about an algorithm used to extract skeletons from natural images. It consists on Learning Multi-task Scale-associated Deep Side Outputs (LMSDS), also known as Deep Skeleton. The reason of its name is because not only tries to extract a single skeleton from a natural image, but also predicts the value of the distance from the closest boundary in each skeleton pixel. We will also use this algorithm to extract skeletons from objects in a natural scene.

The reason why these two models are chosen is due to their architecture similarity as they share the same backbone network. Once both of them are defined, we present our joint

medial axis and boundary detector. This model is capable of extracting edges and skeletons simultaneously using a single convolutional neural network. As we mentioned before, coupled boundary and skeleton extractions share some duality, therefore both tasks of our new model should be consistent with each other. In other words, they should encode the same information with respect to a natural scene. In order to enforce this last aspect, we add an extension that will perform this duty: enforcing consistency between both tasks. This consists on building new boundary and skeleton maps from the previous skeleton and boundary extractions respectively and check the agreement between each other – the new ones with respect to the previous ones.

After creating the joint detector and its extension, we test them in different scenarios and compare their performance with respect to the other detectors (HED for edge detection and LSMDS for skeleton detection). This way, we will be able to see if our hypotheses mentioned previously can be confirmed. Instead of using the traditional data sets where edge and skeleton detection are evaluated, we create a new data set with boundaries and skeletons using natural scenes and annotations from COCO.

1.3 Outline

The rest of the report is structured as follows:

- In chapter 2, we provide an overview of the previous work on edge and skeleton detection in the past years.
- In chapter 3, we provide an overview of fundamental concepts of convolutional neural networks to make the text self-contained.
- In chapter 4, we describe our method for joint learning of boundary and skeleton extraction using convolutional neural networks.
- In chapter 5, we describe all the experiments that have been carried out throughout this project: the process of developing our re-implementations and the performance of our joint medial axis and boundary detector in comparison with them.
- Finally, in chapter 6, we will discuss results and conclude.

Chapter 2

Previous work

2.1 Edge detection

Early approaches framed edge detection in 2D images as a search for local, abrupt changes in image intensity [30], and often relied on simple convolution operations with an appropriate kernel. Such works include the popular Sobel [21] and Canny [8] edge detectors. The Canny edge detector remained for several years the edge detector of choice in a variety of tasks due to its efficiency and robustness against noise. Further information on classical edge detectors can be found in [47] and [7].

With the proliferation of annotated data [31] and the progress in machine learning tools, researchers soon started favoring data-driven approaches over model-based approaches for edge detection. An early example is [22], in which the edge detection task is presented as a statistical inference. In their seminal work, Martin et al. [32], use brightness, color and texture features to train a classifier that produces a probability score of edge existence for every pixel in the image, framing edge detection as a binary classification problem. In a later extension [4] the authors complement the local nature of the features used in [32] by adding global features based on spectral clustering [19], improving performance. The importance of using features at multiple scales has also been recognized [33], while more recent works relied on more powerful classifiers, such as random decision forests, boosting both speed and accuracy [24, 10].

Finally, the modern edge detection algorithms use deep learning approaches, taking advantage of the capability of convolutional neural networks to perform automatic hierarchical feature

learning from natural images. Deep learning approaches include N^4 -Fields [12], DeepContour [43], DeepEdge [6], and Pixel-wise CNN [16]. These algorithms combine convolutional neural networks (CNNs) with several machine learning techniques such as nearest neighbor search [12] and SVM classifiers [16]. They are able to achieve decent results comparable to human performance in edge detection. Perhaps the most well-known work on edge detection using CNNs is the *Holistically Nested Edge Detection (HED)* [45] which uses a Fully Convolutional Networks (FCN) [35] to produce a dense edge prediction output. More recently, a related task called semantic edge detection [28] has been a case of study. It consists on jointly extracting edges as well as their category information within a natural scene.

Our work is more closely related to HED [45]; we extend its architecture so as to build a joint model capable extracting object boundaries and skeletons simultaneously in a natural scene.

2.2 Skeleton detection

Skeleton extraction from natural images is said to be a very challenging task and has been widely studied in recent decades. The early stages of this task focus on detecting symmetries in binary images (pre-segmented shapes) [34]. We can find interesting contributions to skeleton extraction in binary images such as medial axis transform [5] and scale axis transform [13]. They also provide a robust definition to what a skeleton as an encoding structure means. The different skeleton extraction algorithms that have been released can be divided into three groups: early image processing methods, learning-based methods that are based on hand-crafted features and recent CNN architectures. As can be seen, they are structured in a similar way as the edge detection task – recall previous section.

The early image processing methods treat skeleton detection as a morphological operation [34]. All of them follow a common hypotheses for the skeleton to be the middle point of two parallel boundaries. These methods focus on a previous boundary extraction using gradient intensity maps [26], [41]. Some of them are able to directly extract symmetries using this approach based on the spatial second derivative [29]. Other relevant contributions include graph-based clustering algorithms [23], spatial filters [44]. Recently, classical approaches are still used to perform skeleton extraction and segmentation from natural images [18]. However, these approaches appear to be weak when testing on natural scenes because of the complexity

of finding medial axis points.

One example treats skeleton detection as a per-pixel classification problem computing hand-designed features and later using multiple instance learning to classify its symmetry [40]. In [39], the task is formulated as a regression problem to both achieve skeleton localization and distance learning to the closest skeleton in scale-space. Nonetheless, these first learning-based algorithms are unable to detect object symmetries in complex backgrounds, due to the limited representation capacity of hand-crafted features.

More recently, researchers have taken preference of the computing power of convolutional neural networks and are able to achieve promising results within this task. There exists several examples that combine different strategies based on similar architecture setting. Starting from HED [45] which is able to extract skeletons in a pixel-wise fashion, we find FSDS [37], a network that extends the previous approach by classifying skeleton pixels based on their scale. LMSDS [36] extends from [37] by adding a regressor to perform both skeleton localization and scale regression. Side-output Residual Network [20] combines residual blocks to fit the errors found on each side-output stage. Hi-Fi [46] uses a similar architecture in [37] and introduces a hierarchical feature learning mechanism. Linear Span Network [27] introduces linear span units to improve the efficiency of feature integration. Finally, DeepFlux [42] differs from the last methods by introducing the concept of content flux in skeletons, which is the learning objective instead of treating the problem as a binary classification problem.

Chapter 3

Overview of Convolutional Neural Networks

In this chapter we provide an overview of the fundamental concepts related to convolutional neural network architectures and training. We limit our exposition to building blocks used in this project to keep the text self-contained – if the reader is already familiar with these notions, (s)he can skip this chapter. The following explanations have been extracted from [11], [3], [2] and [1].

Before getting into CNNs, let's review the concept of neural networks. Neural networks consists of a set of nodes (also known as neurons) that are grouped into different layers. These layers can be divided in three different groups:

- **Input layer:** Representation of the input forwarded to the network.
- **Hidden layer(s):** Representation of the multiple intermediate states of the network.
- **Output layer:** Representation of the obtained result.

In figure 3.1, we can see a representation of a simple neural network. Each neuron stores a single value and each connection between a neuron and the ones in the previous layer represents products between each value stored in the previous neurons and a parameter (one for each neuron), which is often called weight. Later, another quantity referred as bias is added to the sum of these products and the result is stored in the latter node. It is common to apply a non-linear function (activation function) to the result obtained by the previous operation (fig. 3.2). In fact, these type of operations are the most common in regular

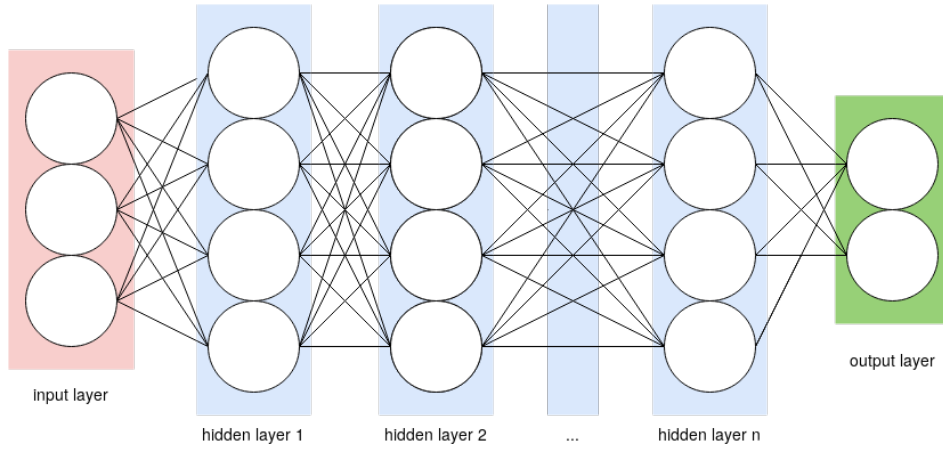


Figure 3.1: Example of a simple neural network consisting of one input layer of 3 neurons, several (n) hidden layers of 4 neurons and an output layer of 2 neurons.

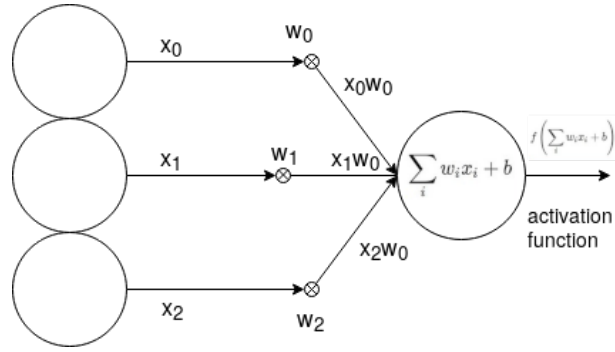


Figure 3.2: Example of a network operation in a fully connected layer. x is the input, w is the weight, b is the bias added to the sum and f is the activation function.

neural networks and are known as fully connected layers. Fully connected layers differ from convolutional ones due to the fact that the first one is connected to all the nodes of the next layer whereas the connectivity of the latter is only local. Our objective is to find the optimal value for each of the weights of the network that fits best to the desired output value.

In a more abstract sense, a neural network can be represented as a function $F(X; \mathbf{W})$, X being the input value and \mathbf{W} the set of weights that the network contains. Therefore, with the correct selection of weights and layer configurations, one can build a neural network that approximates to any other function [1].

However, how do we find the optimal value for the weights? The key of this procedure is to take into account that the layers that form the network are differentiable functions. Having this in mind, we can consider the neural network to be a differentiable function.

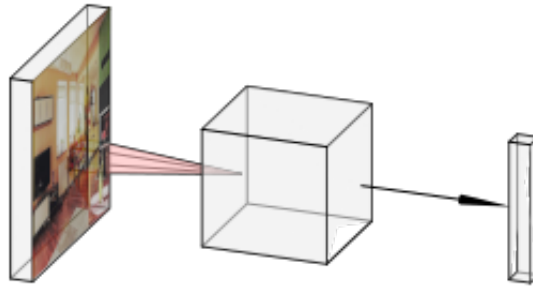


Figure 3.3: Visual example of the 3D arrangement of a CNN.

Consequently, we will be able to define an error function (also differentiable) with respect to the output layer and calculate the gradient of each weight with respect to the error function previously defined (more on section 3.2).

Convolutional neural networks are very similar to regular the neural networks that we have just seen: they are composed by layers with weights and biases that will be optimized to approximate a function. The difference between them is that they assume that the input is an image and they make the function they represent more efficient to implement and reduce drastically the amount of parameters in the network. The name *convolutional* is used because the convolution is the operation which is the most widely used. Furthermore, CNNs take advantage of the way images are represented and have their neurons arranged in three dimensions: height, width and depth. For example, an image of 400x400 pixels will be represented as a three-dimensional array of height 400, width 400 and depth 3 (recall that each pixel of an image has 3 color values, RGB). The term *depth* is also referred as the number of layers of a neural network as well, which is contrary to the meaning that we are giving here.

In the following sections, we will talk about the different types of layers that we have used to build the CNNs for our experiments. Furthermore, we will give an intuition of how the weights are optimized to perform a determined task.

3.1 Layers used in our implementation

As we described above, a convolutional neural network is a sequence of layers. Each of these layers transforms the input they receive through a differentiable function and forward the

output to the next one. It is important to remember that all the functions applied in the different layers that appear in the network are differentiable, otherwise we would not be able to optimize its weights by applying backpropagation.

Below are listed the different types of layers that have been used to build the models that take part in this project: convolution, transposed convolution, non-linear activation, residual block and down-sampling.

3.1.1 Convolution

The convolution is the core layer in a CNN. It consists on a set of learnable filters that extend spatially along its input. The size of each filter is relatively small compared to the size of the input that the layer is received. A typical size for a filter in the very first layer of a CNN could be $3 \times 3 \times 3$ (i.e. 3 pixels wide, 3 pixels high and 3 pixels deep to match with the color channels). The amount of learnable filters that the layer has will determine the depth of the output of the layer. Therefore, when performing a convolution, we may notice three different elements: the input feature map, the output feature map and the kernels or filters.

The forward pass of an input feature map through a convolutional layer is described as follows. We slide (also called convolve) each filter across the width and height of the input feature map in different stages. At each stage, we perform an element-wise product between the overlapping values of the input feature map and the parameters of the filters. The element-wise products performed between each filter and the region of the input feature is summed producing an output value, resulting in a set of values at each stage (one for each filter). In conclusion, each region of the output feature map will be connected to a local region of the input feature map. The spatial extent of this region is known as receptive field size. This term is important when considering which convolutional features one intends to capture.

The configuration of the set of filters consists on a set of hyper-parameters that have to be defined. First we have the depth of the filters, which has to coincide with the depth of the input feature map. Then, we need to determine the amount of filters, or the depth of the output feature map. Moreover, we need to set the width and height of the filters (K_x , K_y). A common setting is to have squared filters (K , same width and height) with the same size. Two other properties that can be configured in a convolutional layer are the stride (S) with which we slide the filters and the size of the zero padding (P). Zero padding means to add

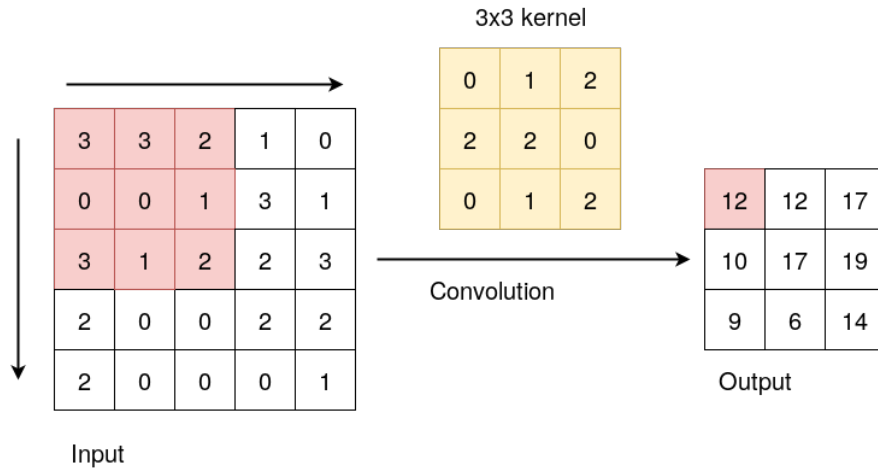


Figure 3.4: Example of a 3x3 convolution with stride 1 and no padding applied. As we can see, the kernel slides along the input feature map, performs a dot product and generates an output feature map as a result.

zero values in the surroundings of the input feature map. If we denote the width of the input feature map as W_{in} , the width of the output feature map (W_{out}) will be:

$$W_{out} = \frac{W_{in} + 2P - K}{S} + 1 \quad (3.1)$$

The receptive field size of the output feature map (R_{out}) depends on the receptive field size of the input feature map (R_{in}), the size of the kernel (K) and the accumulated stride of the input feature map (j_{in}).

$$j_{out} = S \cdot j_{in} \quad (3.2)$$

$$R_{out} = R_{in} + (K + 1) \cdot j_{in} \quad (3.3)$$

This arithmetic is very helpful if one wants to keep track of the properties of a convolutional neural network and can be applied to different types of layers.

In image 3.4 a typical convolution operation is illustrated. For simplicity, the input and output feature maps have one unit of depth. We have considered the parameter sharing setting in this example, therefore only one filter is required. Notice that the receptive field size of the output feature map is 3 as the filter connects locally regions of size 3x3.

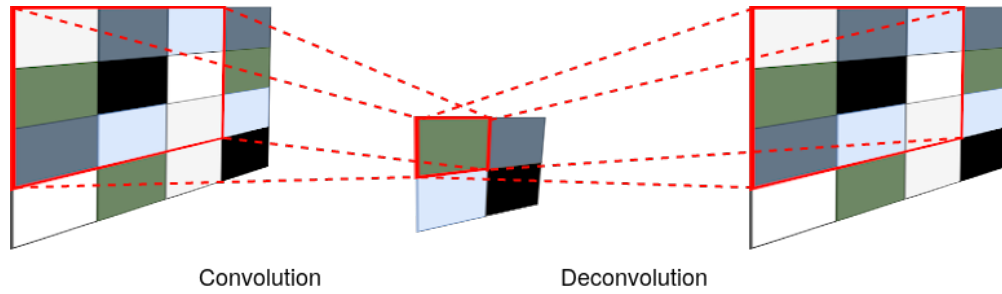


Figure 3.5: Illustration of the comparison between a convolution and a transposed convolution. The latter aims to learn an upsampling kernel to reconstruct the input.

3.1.2 Transposed convolution

The transposed convolution, also called deconvolution or fractionally strided convolution [11], is an layer used to up-sample an input feature map in a learning-based manner. The need for deconvolutions arises from the desire to use a transformation that goes into the opposite direction with respect to the convolution layer. Figure 3.5 describes the relation between these two layers.

To better understand how the transposed convolution works, let's explore the duality with respect convolution with an example. Suppose that we have a 2x2 kernel (W) and a 3x3 input feature map (A) as expressed below.

$$W = \begin{pmatrix} w_0 & w_1 \\ w_2 & w_3 \end{pmatrix}; \quad I = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix}$$

We rearrange the kernel into the the following matrix, denoted as convolution matrix (C) and flatten the input image in a single column matrix (I_f).

$$C = \begin{pmatrix} w_0 & w_1 & 0 & w_2 & w_3 & 0 & 0 & 0 & 0 \\ 0 & w_0 & w_1 & 0 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_0 & w_1 & 0 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & w_0 & w_1 & 0 & w_2 & w_3 \end{pmatrix}$$

$$I_f^T = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{10} & a_{11} & a_{12} & a_{20} & a_{21} & a_{22} \end{pmatrix}$$

The convolution can be therefore expressed as follows.

$$\begin{pmatrix} w_0 & w_1 & 0 & w_2 & w_3 & 0 & 0 & 0 & 0 \\ 0 & w_0 & w_1 & 0 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_0 & w_1 & 0 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & w_0 & w_1 & 0 & w_2 & w_3 \end{pmatrix} \cdot \begin{pmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \\ a_{20} \\ a_{21} \\ a_{22} \end{pmatrix} = \begin{pmatrix} o_{00} & o_{01} & o_{10} & o_{11} \end{pmatrix} = O_f$$

As a result, we obtain a flattened version of the output feature map (O_f). The transposed convolution is equivalent to perform the reverse operation by taking C^T as the convolution matrix. In our implementation, we use this type of layer to up-sample our edge and skeleton maps.

3.1.3 Non-linear activations

It is very common to perform element-wise non-linear operations after each convolutional layer. The most common non-linear activation is known as ReLu, which stands for Rectified Linear Unit for a non-linear operation. Being x and element of the input feature map, the ReLu simply performs $\max(0, x)$. There are other non-linear functions that can be used in a CNN such as tanh and sigmoid. In our architecture, we use the last one to map the activations of the last layer to a $[0, 1]$ interval.

3.1.4 Residual

The residual block is a simple operation that consists on feeding the input feature map to a set of layers – normally convolutions and non-linearities, and adding the result with the original input at the end. It is worth mentioning that the size of the input and output feature maps of all the convolutions within the block must remain invariant, otherwise the latter sum cannot be performed due to dimension mismatching. The advantages of having

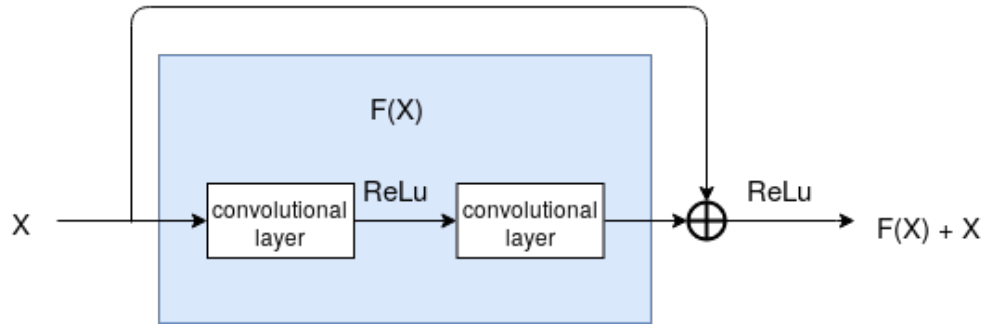


Figure 3.6: Representation of a residual block, where convolutional layers and non-linearities (ReLU) are combined.

residual blocks instead of traditional convolutions is that one can skip the training of several layers using the residual connections.

Figure 3.6 shows an example of the residual block that we use in our architecture setting. We are interested in residual blocks because we will use it to perform domain translations when building image-to-image mapping functions to obtain edge maps from skeleton maps and vice versa (see section 4.4.2).

3.1.5 Pooling

Pooling layers are used mainly to reduce the dimensionality (in terms of height and width) of the input feature map. This layer does not contain any learnable parameters. The procedure of forwarding an input feature map to a down-sampling layer is similar to the convolution. In this case, our filter performs an operations with all the elements of the input feature map that overlap with it and slides with a predetermined stride (S) across the entire width and height.

Some examples of pooling operations are: maxpool (largest of all the elements), averagepool (average of the elements within the filter), sumpool (summation of the all the elements), etc. As an example, in figure 3.7 we show a maxpool operation.

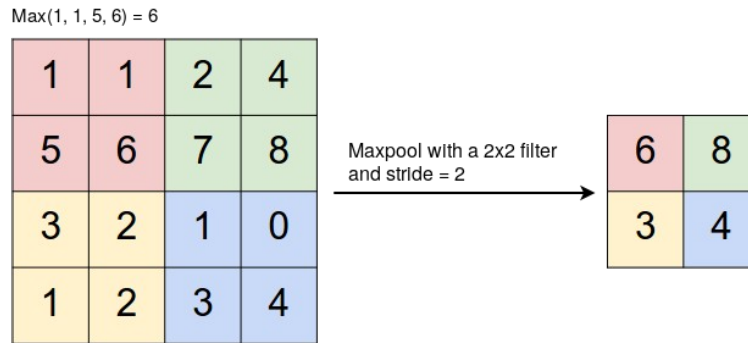


Figure 3.7: Example of a maxpool layer consisting of a 2x2 filter with stride 2.

3.2 Parameter optimization

Recall that previously we mentioned that a neural network (in our case, a convolutional neural network) can be represented as a function $F(X; \mathbf{W})$, X being an input image and \mathbf{W} the parameters of the network. This set of weights includes the learnable parameters of the filters in all the convolutional layers, residual blocks and deconvolutional layers. Our objective is to optimize the value of this set \mathbf{W} so that $F(X; \mathbf{W})$ gives a desired output, such as detecting edges, skeletons, etc. In other words, we aim generalization of a task with respect to new inputs.

In order to optimize the weights we need to define an objective function [3], also called loss function, \mathcal{L} . Theoretically, the role of the loss function is to measure the quality of our network. The idea is to configure the parameters such that we obtain the highest quality possible. To do so, we are given a training set composed by input images paired with their respective ground truth label map. The latter consists on the expected output by means of the input forwarded to the network. The loss function will be in charge of evaluating the agreement of the output of our network with respect to the ground truth label map of the input image. A high value means low agreement, whereas a low value means the opposite.

Once we know the components that take place when designing the response of a neural network, we can talk about optimization: the process of finding the set of weights \mathbf{W} that minimize the loss function. The key of optimization lies in the gradients of the weights, i.e. the partial derivative of the lost function with respect to each weight. This is the reason why all the layers of a neural network must be equivalent to differentiable functions. Otherwise, the optimization procedure cannot be performed. The result of this partial derivative with

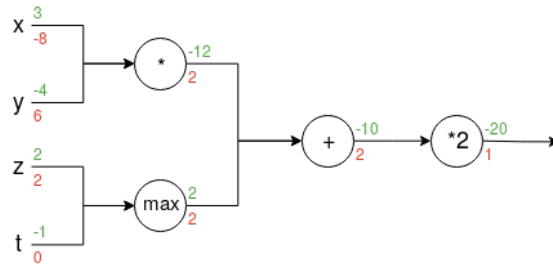


Figure 3.8: Example to understand the intuition of backpropagation. The forward computation is indicated in green, starting at x , y , z and t with values 3, -4 , 2 and -1 respectively. The backward one is indicated in red, starting with 1 at the last step. By performing a recursive derivative chain rule in backwards direction we can compute the gradients of the four input values.

respect to the loss function consists of a set of gradients with the same length as the set of weights. This new set gives us the direction $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$ (expressed as a vector of size $|\mathbf{W}|$) in which we need to move in the space defined by our set of weights so as to minimize the loss function. After this calculation, we will have to update the set of weights as follows:

$$\mathbf{W} \rightarrow \mathbf{W} - h \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{W}} \quad (3.4)$$

where h represents the step that we take when updating the weights towards the direction defined by $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$ in the space of the set of weights, with $|\mathbf{W}|$ dimensions. The procedure of performing this operation throughout the entire training set, one element at a time, is called Stochastic Gradient Descent (SGD). In fact this is a simplified version of SGD, but we wanted to describe a simplify version to ease the explanation of the optimization phase. An epoch, is the process of running the SGD algorithm across the training set once. Usually this is run during several epochs until convergence of the loss function is observed.

In conclusion, for each weight $\{w_i, i = 1, \dots, |\mathbf{W}|\}$ we will have to calculate $\frac{\partial \mathcal{L}}{\partial w_i}$. Considering that our network is a set of millions of parameters and element-wise operations, this is practically impossible. However, the gradient of the weights is calculated using a method called backpropagation. It consists of recursively applying the chain rule along the network operations. In order to understand how this works, an example can be observed in figure 3.8. Recall that convolutions, downsamplings, etc, are composed by products, sums or other simple differentiable operations, therefore we can numerically evaluate the gradient of the entire network using backpropagation so as to perform parameter optimization.

Chapter 4

Joint learning of object boundaries and skeletons

In this chapter we will present and explain the different architectures that have been used in our experiments. As a reminder, we are first re-implementing the Holistically-nested Edge Detection and Deep Skeleton algorithms. We are using these two architectures in order to compare their performance with respect to the extensions that we are planning to build in this report. Next, we will take advantage of the similarity of the first two architectures and combine them to address the two tasks at the same time. Moreover, we will extend the last model by adding an image translation approach so as to provide consistency between both tasks.

4.1 Holistically-nested Edge Detection

Holistically-nested edge detection (HED) [45] is described as an end-to-end edge detection system. It automatically learns the type of hierarchical features within a natural image that are key to imitate the human ability to resolve the ambiguity in edge and boundary extraction. The reason why the authors of the algorithm use "holistic" is because the detector aims to output the edge prediction in an image-to-image fashion. This will be possible using a common network architecture known as Fully Convolutional Network (FCN) [35]. On the other hand, the word "nested" means that we will be extracting different edge predictions and producing them as side outputs (SO) throughout the procedure of boundary extraction.

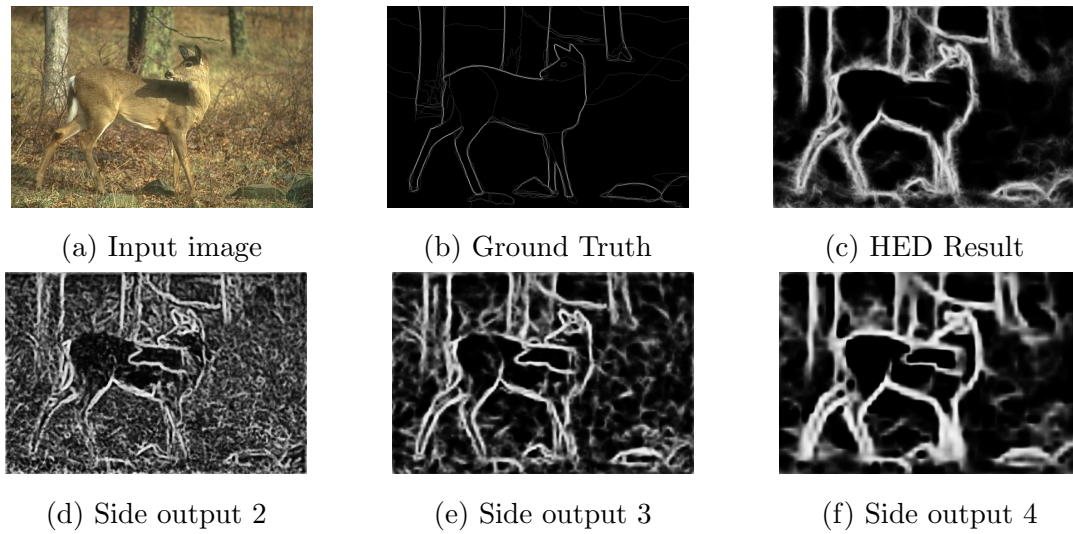


Figure 4.1: Example of the HED algorithm response. (a) shows an example image from BSDS500 [4]. (b) shows its corresponding edges annotated by humans. (c) corresponds to the HED output. (d), (e), and (f) show the side-output responses at different stages.

We will see that as we go deeper, these side outputs will provide more refined predictions as they will have more information of the global features that define the detected edges. Figure 4.1 shows an illustrative example of the prediction refinement and the global features detected in deeper levels. The reason why this is possible lies on the receptive field size of the network used, which grows at each side output prediction.

In figure 4.2, we can see a representation of the holistically-nested network architecture. As we can see, the multiple side outputs that are produced in the detection give us the capability of adding a final output layer which will take into account the predictions in the different levels of feature extraction.

4.1.1 Formulation

Now we are going to explain the formulation of the approach for this edge detector.

Training Phase

Let us consider that we have a training set which is denoted by $\{(X_n, Y_n), n = 1, \dots, N\}$, where $X_n = \{x_j^{(n)}, j = 1, \dots, |X_n|\}$ represents an input image and $Y_n = \{y_j^{(n)}, j = 1, \dots, |X_n|\}$,

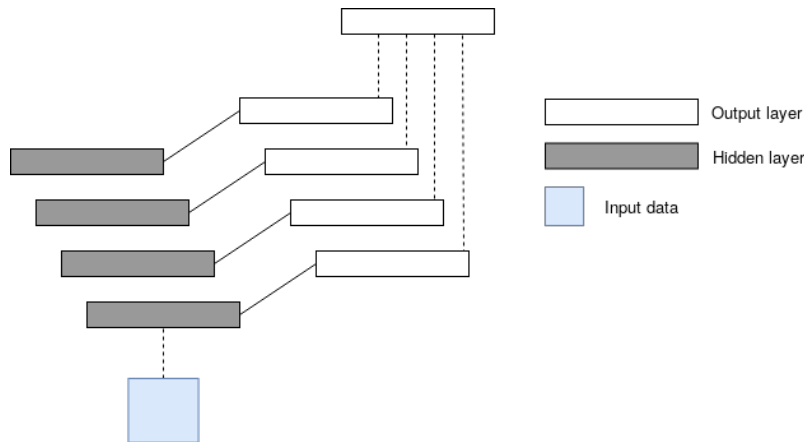


Figure 4.2: Holistically-nested network architecture.

$y_j^{(n)} \in \{0, 1\}$ represents the corresponding ground truth binary edge map for image X_n , both consisting of $|X_n|$ pixels. For each pixel, its corresponding ground truth binary map will label non-edge pixels as 0 and edge pixels as 1. From here on out, we will be considering just one image and we will drop the subscript n for simplicity.

Our objective is that the network learns the features within the image so as to produce an edge map closest to the ground truth. Suppose that this network consists of a collection of layers in which its parameters are represented as \mathbf{W} . Moreover, we have M side-output layers associated with a classifier according to the architecture described before. We denote the weights of these last ones as $\mathbf{w} = (\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(M)})$.

Consider the function

$$\mathcal{L}_{side}(\mathbf{W}, \mathbf{w}) = \sum_{m=1}^M \alpha_m \ell_{side}^{(m)}(\mathbf{W}, \mathbf{w}^{(m)}), \quad (4.1)$$

being $\ell_{side}^{(m)}(\mathbf{W}, \mathbf{w}^{(m)})$ the loss function at each side output and α_m a hyper-parameter for each side-output loss term.

In our training, the loss function will be computed over all the pixels of the input image $X = \{x_j, j = 1, \dots, |X|\}$ and the ground truth edge map $Y = \{y_j, j = 1, \dots, |X|\}, y_j \in \{0, 1\}$. It is very common for a natural image that its edge prediction map is heavily biased: almost all ground truth pixels are non-edge. Therefore, the authors adopt a loss function to automatically balance the loss contributions of edge/non-edge pixels. They introduce a class-balancing weight β to offset the imbalance between both classes (edge/non-edge). Considering this, the previous $\ell_{side}^{(m)}(\mathbf{W}, \mathbf{w}^{(m)})$ is defined as a class-balanced cross-entropy

function as follows.

$$\ell_{side}^{(m)}(\mathbf{W}, \mathbf{w}^{(m)}) = -\beta \sum_{j \in Y_+} \log Pr(y_j = 1|X; \mathbf{W}, \mathbf{w}^{(m)}) - (1 - \beta) \sum_{j \in Y_-} \log Pr(y_j = 0|X; \mathbf{W}, \mathbf{w}^{(m)}) \quad (4.2)$$

where $\beta = |Y_-|/|Y|$ and $1 - \beta = |Y_+|/|Y|$. $|Y_-|$ and $|Y_+|$ represent the non-edge and edge ground truth label sets respectively. $Pr(y_j = 1|X; \mathbf{W}, \mathbf{w}^{(m)}) = \sigma(a_j^{(m)}) \in [0, 1]$ is computed using the sigmoid function $\sigma(\cdot)$ on the value at each pixel j .

Each side output layer m will produce an edge map prediction $\hat{Y}_{side}^{(m)} = \sigma(\hat{A}_{side}^{(m)})$, where $\hat{A}_{side}^{(m)} = \{a_j^{(m)}, j = 1, \dots, |X|\}$ are the activations of the side-output layer. According to the representation mentioned before and, to directly take advantage of the side-output predictions, a "weighted fusion" layer is added. The weights for this fusion will be learned during training and its loss function is defined as

$$\mathcal{L}_{fuse}(\mathbf{W}, \mathbf{w}, \mathbf{h}) = Dist(Y, \hat{Y}_{fuse}) \quad (4.3)$$

where $\hat{Y}_{fuse} = \sigma(\sum_{m=1}^M h_m \hat{A}_{side}^{(m)})$ and $\mathbf{h} = (h_1, \dots, h_m)$ is the fusion weight. $Dist(\cdot, \cdot)$ is a distance function between the ground truth map and the fused predictions, which is set to be a balanced cross-entropy loss as defined in Equation 4.2.

If we put the previous definitions together, the training procedure will be minimizing the following objective function over the entire set using standard stochastic gradient descent (recall Section 3.2):

$$(\mathbf{W}, \mathbf{w}, \mathbf{h})^* = argmin(\mathcal{L}_{side}(\mathbf{W}, \mathbf{w}) + \mathcal{L}_{fuse}(\mathbf{W}, \mathbf{w}, \mathbf{h})) \quad (4.4)$$

Testing Phase

If we forward an image X to the network (denoted below as $CNN(X, (\mathbf{W}, \mathbf{w}, \mathbf{h})^*)$), it outputs an edge map for each side output layer and a fused edge map combining all the previous ones.

$$(\hat{Y}_{fuse}, \hat{Y}_{side}^{(1)}, \dots, \hat{Y}_{side}^{(M)}) = CNN(X, (\mathbf{W}, \mathbf{w}, \mathbf{h})^*) \quad (4.5)$$

The final output that will be taken into account will be the one produced by the weighted fusion layer.

$$\hat{Y}_{HED} = \hat{Y}_{fuse} \quad (4.6)$$

This is contrary to the original implementation which takes as final output an average of all the outputs. The reason why we choose a different final result is because further experiments of the same authors support that this choice improves performance.

4.1.2 Implementation

The choice of the architecture used to implement the approach described above needs two main properties. First, it has to be deep enough to efficiently generate multi-level features. Moreover, it should have multiple stages with different strides to be able to capture edges at different scales. To fit these two properties, the authors propose VGG16 [38] to be the backbone network of the model. VGG16 consists of a network with great depth (16 convolutional layers), great density (stride-1 convolutional kernels), and multiple stages (five stride-2 downsampling layers). In table 4.1, we can see the several layers that compose this network and the different convolutional stages in which is organized.

Therefore, this architecture with the following modifications is adopted to implement HED:

- A side output layer is connected to the last convolutional layer in each stage, i.e. before each downsampling layer (see figure 4.3).
- The last stage of the network is cut (last downsampling layer and the fully connected layers).
- Downsampling the image will reduce its size, therefore an upsampling layer at the end of each side output layer is needed to match the size of the input.

By applying this modifications, the final architecture will have 5 side outputs with different receptive field sizes. Therefore, they will be able to capture edges at different scales, all nested in the same VGG16 network. In figure 4.3 we have sketched a representation of the HED architecture.

In section 5.3.1, we describe the process of re-implementing this algorithm and compare its performance with the original implementation.

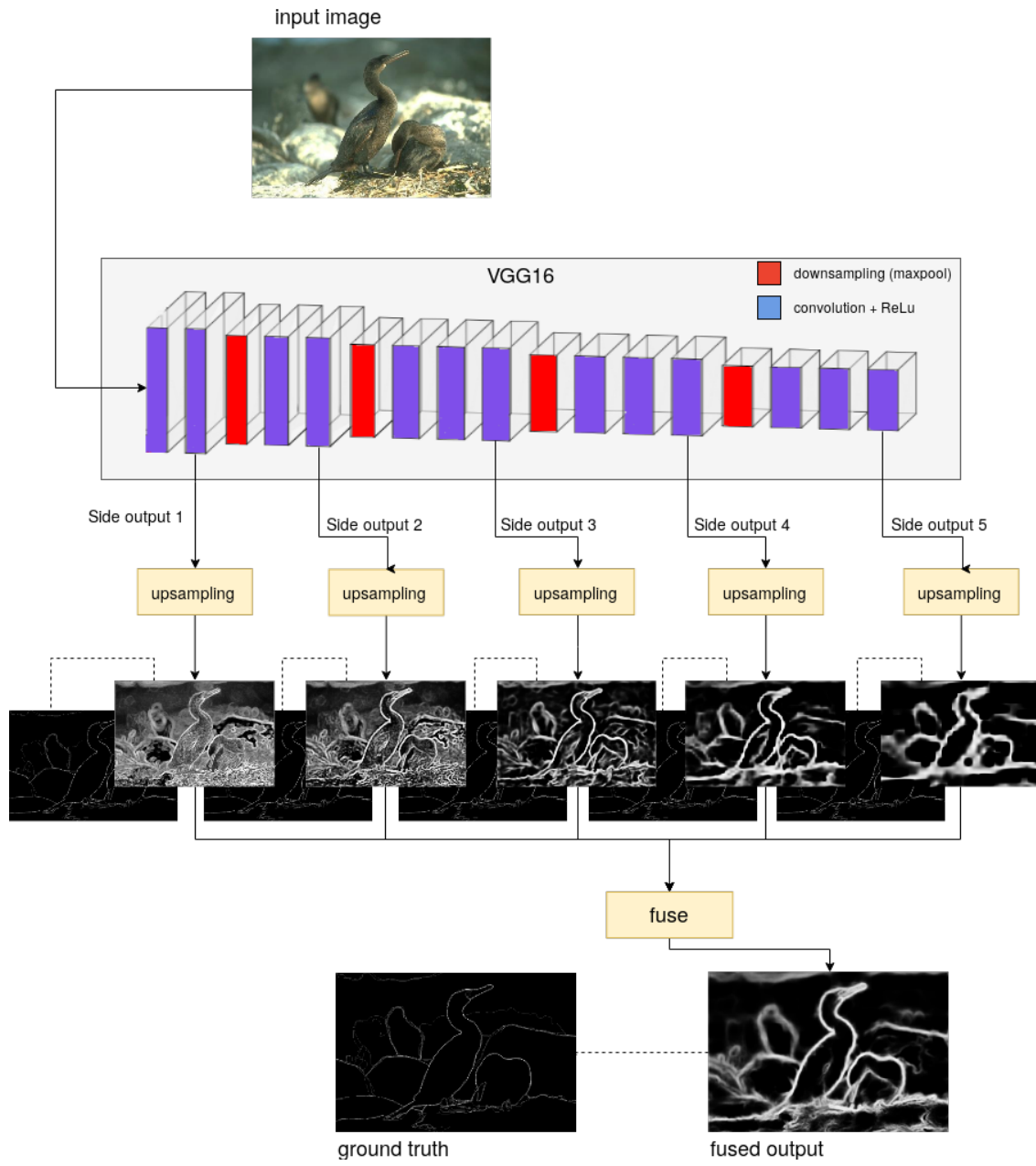


Figure 4.3: Illustration of the HED architecture implementation. One upsampling layer is added after each side output layer to match the size of the input image. The fusion layer learns how to combine responses from multiple scales automatically. The dashed lines at each side output indicate supervision performed by $\ell_{side}^{(1)}$, $\ell_{side}^{(2)}$, $\ell_{side}^{(3)}$, $\ell_{side}^{(4)}$ and $\ell_{side}^{(5)}$ respectively. The one at the fusion layer level corresponds to \mathcal{L}_{fuse} .

block num.	operation	stride	depth	receptive field size
0	input	1	3	1
1	3x3conv	1	64	3
1	3x3conv	1	64	5
2	maxpool	2	64	6
2	3x3conv	2	128	10
2	3x3conv	2	128	14
3	maxpool	4	128	16
3	3x3conv	4	256	24
3	3x3conv	4	256	32
3	3x3conv	4	256	40
4	maxpool	8	256	44
4	3x3conv	8	512	60
4	3x3conv	8	512	76
4	3x3conv	8	512	92
5	maxpool	16	512	100
5	3x3conv	16	512	124
5	3x3conv	16	512	156
5	3x3conv	16	512	196

Table 4.1: Description of the layers that compose VGG16 [38]. The receptive field sizes in bold are the ones of each side output.

4.2 Deep Skeleton

The Deep Skeleton algorithm [37], [36] consists of an extension of the holistically-nested edge detection previously mentioned. The difference between them is that Deep Skeleton performs two tasks at each side output: pixel classification and scale regression. In other words, not only tries to predict if a pixel is a skeleton of an image (binary classification, as in edge detection), but also tries to predict its scale. A scale of a skeleton pixel is defined as the distance between the skeleton pixel and the closest boundary of the object. Contrary to the last procedure, we will not classify an image pixel as skeleton/non-skeleton. In this case, the skeleton pixels will be arranged in different classes depending on its scale (multiclass classification). The reason why the authors decide to address this task using this approach lies on the features that the network is capable of extracting at each stage. The capability of the network to detect an skeleton pixel depends on the receptive field size at a given stage. In figure 4.4, we give an example to the previous explanation. Having this in mind, at each side output the network will only be able to detect skeleton pixels with scales smaller than

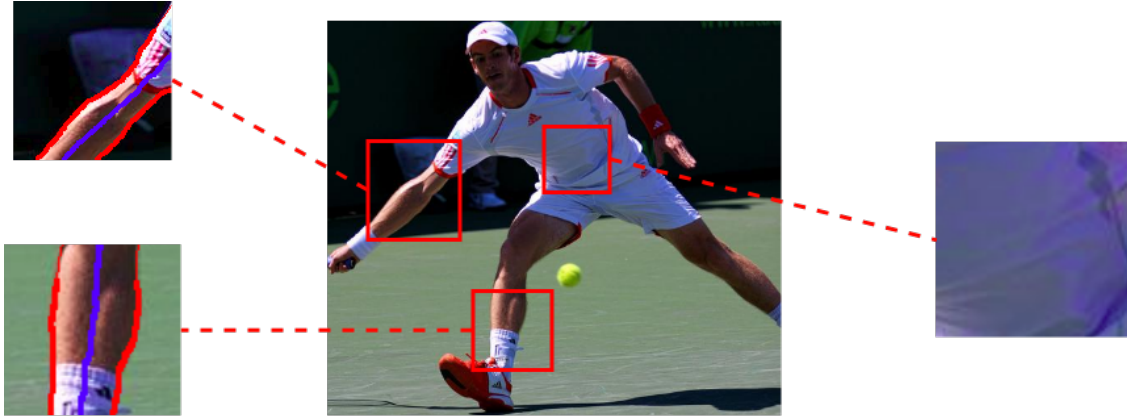


Figure 4.4: Example of the skeleton extraction at different parts of an object for different filter sizes. If we are able to capture the boundaries of the object, we can extract its medial axis points. In our network, the receptive field size increases at each stage, and therefore, the range for skeleton extraction.

their receptive field size. Consequently, the ground truth labels used at each side output will need to be modified in order to be coherent with the skeleton predictions at each stage. The labels with larger scale than the receptive field size will be considered as non-skeleton pixels, as they cannot be detected. For this reason, the side outputs become scale-associated; as we need a different ground truth label at each stage.

4.2.1 Formulation

Now, we will explain the formulation for the deep skeleton algorithm used to extract skeletons from natural images. Recall that contrary to the edge detection approach, our goal here is to address two tasks: skeleton localization and scale prediction.

Training Phase

Suppose that we are given a training set denoted by $\{(X_n, Y_n, S_n), n = 1, \dots, N\}$, where $X_n = \{x_j^{(n)}, j = 1, \dots, |X_n|\}$ represents an input image and $Y_n = \{y_j^{(n)}, j = 1, \dots, |X_n|\}$ ($y_j^{(n)} \in \{0, 1\}$) and $S_n = \{s_j^{(n)}, j = 1, \dots, |X_n|\}$ ($s_j^{(n)} \geq 0$) its corresponding ground truth skeleton and scale map respectively. Notice that $y_j^{(n)} = \mathbf{1}(s_j^{(n)} > 0)$, where $\mathbf{1}(\cdot)$ is the indicator function. A scale value s_j takes integer values and represents the distance to the

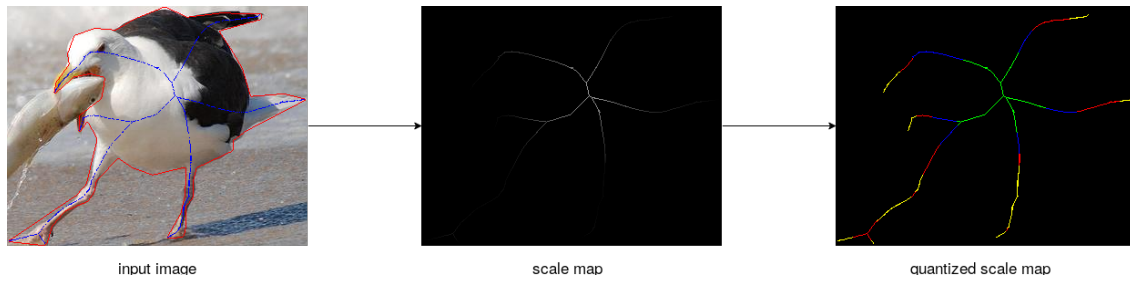


Figure 4.5: Example of the generation of the scale map Z . The input image X has a corresponding ground truth skeleton map Y (first image in blue) and scale map S (middle image). The values in which the quantized scale map Z can be labeled are represented different colors, which refer to different ranges of scales.

closest object boundary expressed in pixels. We drop the superscript n for simplicity as we will consider one image at a time.

As we explained before, we are not just distinguishing skeleton/non-skeleton pixels, but also the scale of them. Therefore, we need to create a ground truth skeleton map to organize the information of the scales in a multi-class fashion. The authors refer to this new ground truth map as the quantized skeleton scale map.

Let us assume that we have M stages in our network. Each of this stages will be linked with two scale-associated side output (SSO) layers. One is in charge of the skeleton localization, the other predicts the scale. We define as $(r_i, i = 1, \dots, M)$ the sequence of the receptive field sizes of the different stages. Moreover, we define z as the quantized scale of a skeleton pixel.

$$z = \begin{cases} \arg \min_{i=1, \dots, M} i, \text{ s.t. } r_i > \rho s & \text{if } s > 0 \\ 0 & \text{if } s = 0 \end{cases} \quad (4.7)$$

where $\rho > 1$ is a hyper parameter to ensure that the receptive field sizes are large enough. Same as the authors did, we set $\rho = 1.2$ in our experiments.

By applying the quantize procedure to each element of the scale map S , we obtain a quantized scale map $Z = \{z_j, j = 1, \dots, |X|\}$ ($z_j \in \{0, 1, \dots, M\}$), in which each number refers to a scale range (example in figure 4.5).

a) Skeleton pixel classification

Once we have our quantized scale map, we can guide the network training for skeleton

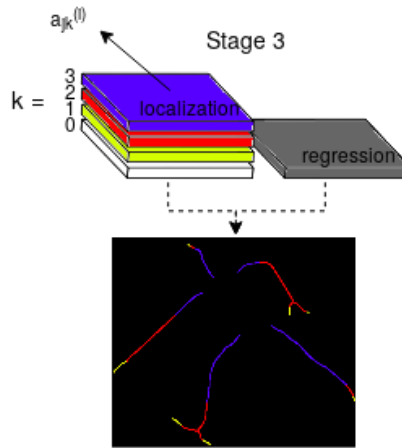


Figure 4.6: Example of the computation of the scale-associated side outputs. Here, stage 3 is shown. Thus, $i = 3$. $a_{jk}^{(i)}$ represent the activations of the skeleton localization side output.

The ground truth skeleton map represents the quantized scale map at this stage $Z^{(i)}$

localization with Z . According to what we discussed before, the network has the capability to detect the skeleton pixels with scales lower than its receptive field size – i.e., the side outputs are scale-associated. Therefore, we have to adapt the ground truth map to each stage of the network as follows. For each stage i , we build a scale-associated quantized scale map: $Z^{(i)} = Z \circ \mathbf{1}(Z \leq i)$, where \circ is an element-wise product operator. $Z^{(i)} = \{z_j^{(i)}, j = 1, \dots, |X|\}$ ($z_j^{(i)} \in \{0, 1, \dots, i\}$). We show an example in figure 4.6.

After each skeleton localization layer, we define a loss function $\ell_{cls}^{(i)}(\mathbf{W}, \Phi^{(i)})$ which is computed over all pixels of the input image X and the scale-associated ground truth map $Z^{(i)}$ – \mathbf{W} represents the weights of the network and $\Phi^{(i)}$ the ones in the i -th side-output layer related to skeleton localization. Similar to the edge detection algorithm, the labels on a skeleton map are also heavily biased towards non-skeleton pixels. Therefore, the authors define a weighted softmax loss function to balance the loss between the different classes:

$$\ell_{cls}^{(i)}(\mathbf{W}, \Phi^{(i)}) = -\frac{1}{|X|} \sum_{j=1}^{|X|} \sum_{k=1}^i \beta_k^{(i)} \mathbf{1}(z_j^{(i)} == k) \log \Pr(z_j^{(i)} = k | X; \mathbf{W}, \Phi^{(i)}) \quad (4.8)$$

where $\beta_k^{(i)}$ is the loss weight for each class k and $\Pr(z_j^{(i)} = k | X; \mathbf{W}, \Phi^{(i)}) \in [0, 1]$ is the predicted score for how likely the quantized scale of x_j is k . We define the class balancing weights $\beta_k^{(i)}$ as follows.

$$\beta_k^{(i)} = \frac{\frac{1}{\mathcal{N}(\mathbf{1}(Z^{(i)}=k))}}{\sum_{l=0}^i \frac{1}{\mathcal{N}(\mathbf{1}(Z^{(i)}=l))}}, \quad (4.9)$$

where $\mathcal{N}(\cdot)$ is defined as the number of non-zero elements in a set. Let $a_{jk}^{(i)}$ represent the activation of the i -th side-output layer associated with the quantized scale k for the input x_j . Then $Pr(z_j^{(i)} = k|X; \mathbf{W}, \Phi^{(i)})$ is computed using the softmax function over each class k . We use $\theta(\cdot)$ to represent the softmax function below.

$$Pr(z_j^{(i)} = k|X; \mathbf{W}, \Phi^{(i)}) = \theta(a_{jk}^{(i)}) = \frac{\exp(a_{jk}^{(i)})}{\sum_{l=0}^i \exp(a_{jl}^{(i)})} \quad (4.10)$$

b) *Skeleton scale prediction*

We now describe the loss function used for scale regression. As we know, we have to adapt the ground truth scale map due to the receptive field size at each stage. Therefore, we will use the value of the receptive field size of the current stage as a reference for scale normalization and our loss term will take into account the following normalized scale-associated ground truth map $\bar{S}^{(i)} = 2\frac{Z^{(i)} \circ S}{r_i} - 1$. This computation maps each scale value s_j into the range $[-1, 1)$. Each side output layer will provide an activation denoted as $\hat{s}_j^{(i)}$ for the input x_j . The loss function of the scale regression is defined as a regression loss based on euclidean distance as follows:

$$\ell_{reg}^{(i)}(\mathbf{W}, \Psi^{(i)}) = \frac{\sum_{j=1}^{|X|} \mathbf{1}(z_j^{(i)} > 0) \|\hat{s}_j^{(i)} - \bar{s}_j^{(i)}\|_2^2}{\mathcal{N}(\mathbf{1}(Z^{(i)} > 0))} \quad (4.11)$$

where $\Psi^{(i)}$ represents the weights of the scale regression layer at the i -th stage. As we can see, the non-skeleton pixels and the ones with scales higher than the receptive field size of the i -th stage are not taken into account – notice that the scale-associated quantized value for each pixel $z_j^{(i)}$ is used.

c) *Multi-task loss*

As we have seen, each side-output provides two loss terms. Hence we define a multi-task loss to train both regression and localization tasks at the same time.

$$\ell_s^{(i)}(\mathbf{W}, \Phi^{(i)}, \Psi^{(i)}) = \ell_{cls}^{(i)}(\mathbf{W}, \Phi^{(i)}) + \lambda \ell_{reg}^{(i)}(\mathbf{W}, \Psi^{(i)}) \quad (4.12)$$

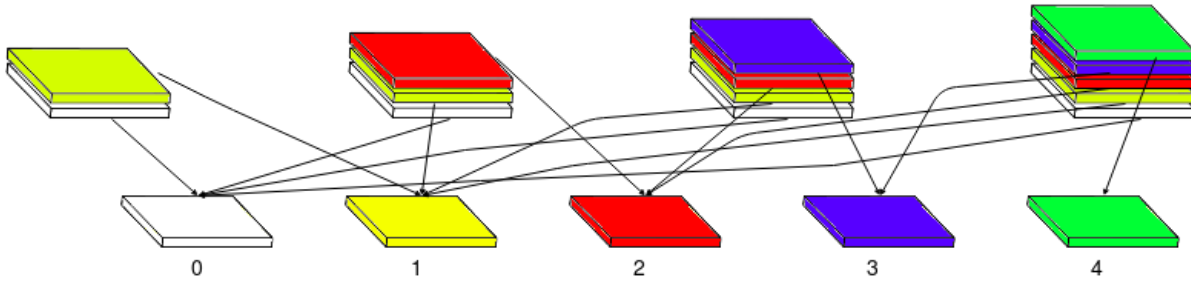


Figure 4.7: Illustration of the fusion of a LMSDS network with 4 stages.

where λ is a hyper-parameter that controls the trade-off between both terms. For all the side output layers, we have the following loss function.

$$\mathcal{L}_s(\mathbf{W}, \Phi, \Psi) = \sum_{i=1}^M \ell_s^{(i)}(\mathbf{W}, \Phi^{(i)}, \Psi^{(i)}) \quad (4.13)$$

The weights of the skeleton localization and regression layer are denoted as $\Phi = (\Phi^{(0)}, \dots, \Phi^{(M)})$ and $\Psi = (\Psi^{(0)}, \dots, \Psi^{(M)})$, respectively.

d) *Scale-associated side outputs fusion*

Consider a pixel x_j of the input image X . Each scale-associated side output provides a score $Pr(z_j^{(i)} = k | X; \mathbf{W}, \Phi^{(i)})$ that represents the likelihood of x_j quantized as k – as we know from previous explanations. We can combine the results of each side-output and obtain a fused score f_{jk} for each class using a weighted sum.

$$f_{jk} = \sum_{i=\max(k,1)}^M h_k^{(i)} Pr(z_j^{(i)} = k | X; \mathbf{W}, \Phi^{(i)}) \quad (4.14)$$

For each class k , we will have a set of weights $\mathbf{h}_k = (h_k^{(i)}; i = \max(k, 1), \dots, M)$. Notice that the size of the set of weights decreases as we move to the higher classes. The intuition behind this fusion layer is the following. Each side output layer provides scores associated to the different classes within the stage. The fusion layer combines the scores for each class obtained at each stage and generates a new score f_{jk} . Then, our fusion layer consists of a set of $M+1$ weight layers (one for each class k) denoted as $\mathbf{H} = (\mathbf{h}_k, k = 0, \dots, M)$. Figure 4.7 illustrates the fusion step. Finally, we define the

fusion loss function as follows:

$$\mathcal{L}_f(\mathbf{W}, \Phi, \mathbf{H}) = -\frac{1}{|X|} \sum_{j=1}^{|X|} \sum_{k=1}^i \beta_k^{(i)} \mathbf{1}(z_j^{(i)} == k) \log Pr(z_j^{(i)} = k | X; \mathbf{W}, \Phi, \mathbf{h}_k) \quad (4.15)$$

where β_k is defined in eqn. 4.9 and $Pr(z_j^{(i)} = k | X; \mathbf{W}, \Phi, \mathbf{h}_k) = \theta(f_{jk})$ is computed as in eqn. 4.10.

As we did in the previous section, we group all the loss terms and minimize the following objective function using standard stochastic gradient descent.

$$(\mathbf{W}, \Phi, \Psi, \mathbf{H})^* = \operatorname{argmin}(\mathcal{L}_s(\mathbf{W}, \Phi, \Psi) + \mathcal{L}_f(\mathbf{W}, \Phi, \mathbf{h})) \quad (4.16)$$

This formulation is what gives name to this algorithm: LMSDS, which stands for Learning Multi-task Scale-associated Deep Side-outputs.

Testing Phase

Given our network with the set of learned parameters $(\mathbf{W}, \Phi, \Psi, \mathbf{H})^*$, a testing image $X = \{x_j, j = 1, \dots, |X|\}$ will provide a predicted skeleton map $\hat{Y} = \{\hat{y}_j, j = 1, \dots, |X|\}$ obtained by

$$\hat{y}_j = 1 - Pr(z_j = 0 | X; (\mathbf{W}, \Phi, \mathbf{h}_0)^*) \quad (4.17)$$

considering that $z_j = 0$ means that the pixel is non-skeleton. Moreover, we can calculate a predicted scale map $\hat{S} = \{\hat{s}_j, j = 1, \dots, |X|\}$. First, we need the value of the most-likely quantized scale value for each pixel which we denote as \hat{i}_j .

$$\hat{i}_j = \operatorname{arg} \max_{i=(1, \dots, M)} Pr(z_j = i | X; (\mathbf{W}, \Phi, \mathbf{h}_i)^*) \quad (4.18)$$

Once we have the value of the quantized scale for each pixel, we need the activation of the \hat{i}_j -th stage of the scale regression step $\hat{\hat{s}}_j^{(\hat{i}_j)}$ and map it to its original value.

$$\hat{s}_j = \frac{\hat{\hat{s}}_j^{(\hat{i}_j)} + 1}{2} r_{\hat{i}_j} \quad (4.19)$$

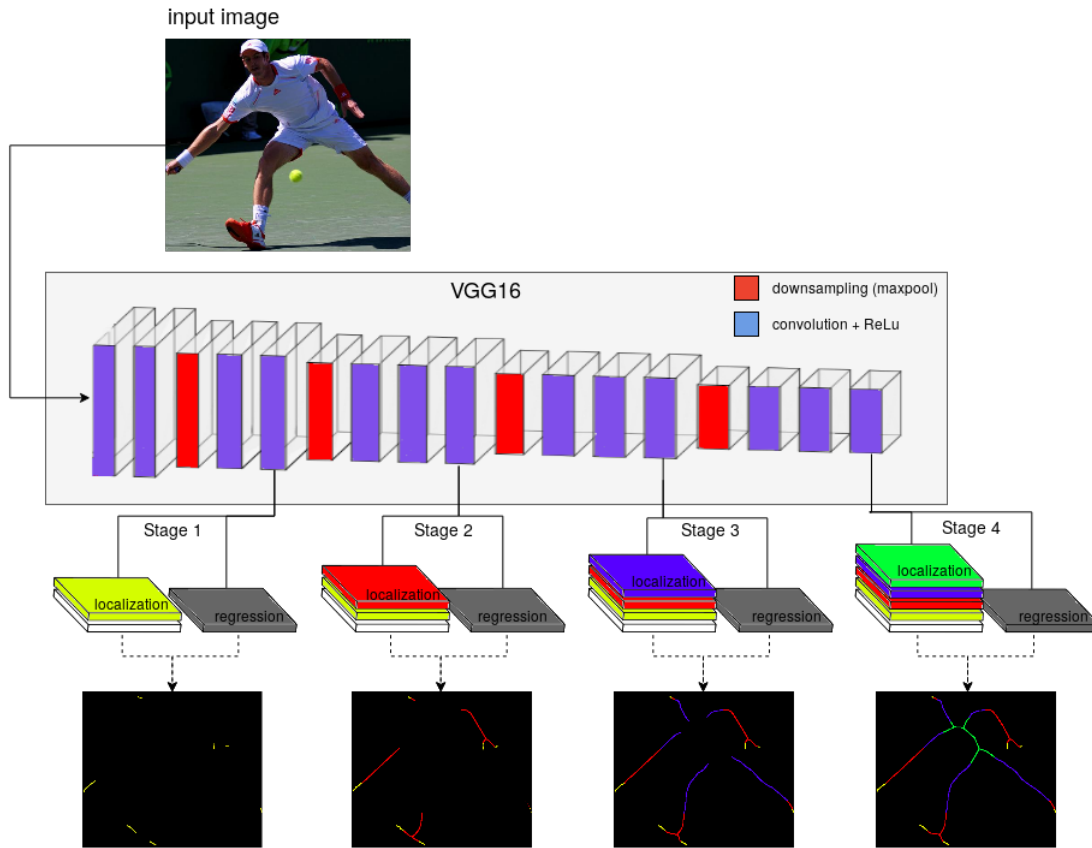


Figure 4.8: Illustration of the LMSDS architecture implementation. As we mentioned, VGG16 is the backbone network of the skeleton detector, which results in 4 different stages.

Each stage adds a quantized scale label and we perform supervision for both skeleton localization and scale regression with respect to the quantized ground truth scale map. We must not forget about the fusion layer (figure 4.7) and its corresponding supervision (not represented here).

4.2.2 Implementation

The choice of the architecture to implement such formulation has to have the same properties as the one used in edge detection. Therefore, the authors decide to adapt HED architecture to build the skeleton detector. An illustration of the LMSDS architecture is shown in figure 4.8.

Starting from the edge detector architecture that we have seen previously, two main modifications are made.

- Instead of one layer, two sibling side-output layers to the last convolutional layer each

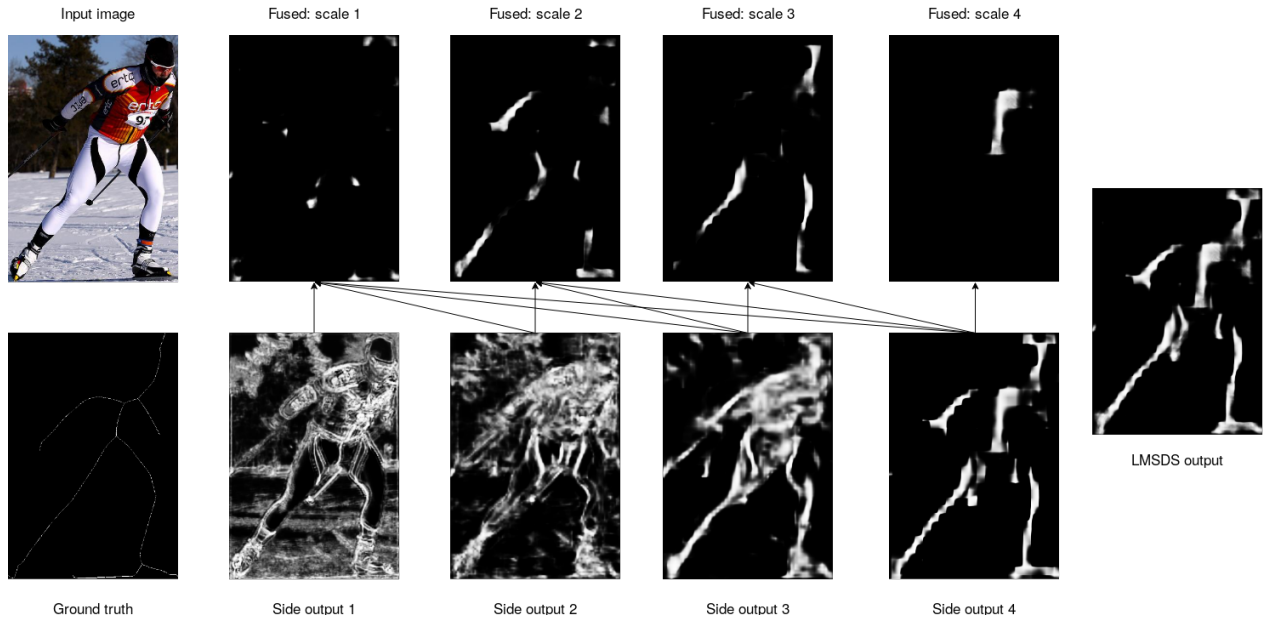


Figure 4.9: Illustrative example of the skeleton localization responses at each stage. The fusion output is also represented and arranges the side output classification to provide a global result.

stage of the VGG16 network are connected. One layer is used for quantized scale classification and the other for scale regression.

- Each layer that performs the quantized scale classification is then sliced to obtain scores at each class and stage. Then the scores of the different stages are fused as described in the formulation.

VGG16 has five convolutional stages. The receptive field sizes at each stage are 5, 14, 40, 92, 196, respectively (recall table 4.1). The authors decide to omit the first stage because its receptive field size is too small to capture any medial axis features. As an example, in figure 4.9 we can see the responses of the skeleton localization at the different quantized scales.

To sum up, this network consists of 4 stages (one less than in HED). Therefore, the scales of the skeleton are classified into 4 different quantized labels groups following the definition in eqn. 4.7.

4.3 Joint edge and skeleton detector

At this point, the reader can regard that we have two different convolutional neural networks:

- **HED:** Model in charge of detecting edges in an image. It addresses the edge detection task.
- **LMSDS:** Model in charge of detecting medial axis points. It addresses the skeleton detection task.

Having this in mind, we will create a new model that performs edge and skeleton detection simultaneously. This model consists of two extensions. Taking advantage of the architecture similarity of HED and LMSDS, we build the first extension which has the same backbone network for both tasks. The second one tries to exploit the duality between edges and skeletons by adding a consistency loss term to the objective function using the image-to-image mapping functions that we have developed before.

4.3.1 Formulation

Training Phase

Let us define the objective function of the first extension of the joint network. It consists on a multi-task loss to provide supervision for edge and skeleton detection simultaneously. We are supposing our network has a shared backbone network, which parameters are denoted by \mathbf{W} . Recall the objective function of the edge detector (eqn. 4.4).

$$\mathcal{L}_{edge}(\mathbf{W}, \mathbf{w}, \mathbf{h}) = \mathcal{L}_{side}(\mathbf{W}, \mathbf{w}) + \mathcal{L}_{fuse}(\mathbf{W}, \mathbf{w}, \mathbf{h}) \quad (4.20)$$

Moreover, recall the objective function of the skeleton detector (eqn. 4.16)

$$\mathcal{L}_{ske}(\mathbf{W}, \Phi, \Psi, \mathbf{H}) = \mathcal{L}_{cls}(\mathbf{W}, \Phi) + \mathcal{L}_f(\mathbf{W}, \Phi, \mathbf{H}) + \lambda \mathcal{L}_{reg}(\mathbf{W}, \Psi) \quad (4.21)$$

where the loss terms have been rearranged – $\mathcal{L}_{cls}(\mathbf{W}, \Phi) = \sum_{i=1}^M \ell_{cls}^{(i)}(\mathbf{W}, \Phi^{(i)})$ and $\mathcal{L}_{reg}(\mathbf{W}, \Psi) = \sum_{i=1}^M \ell_{reg}^{(i)}(\mathbf{W}, \Psi^{(i)})$. Eventually, we obtain the loss term that defines the objective of the joint boundary and skeleton detector.

$$(\mathbf{W}, \mathbf{w}, \mathbf{h}, \Phi, \Psi, \mathbf{H})^* = \arg \min(\mathcal{L}_{edge}(\mathbf{W}, \mathbf{w}, \mathbf{h}) + \gamma \mathcal{L}_{ske}(\mathbf{W}, \Phi, \Psi, \mathbf{H})), \quad (4.22)$$

where γ is a hyper-parameter that balances the contributions for both tasks. To sum up, we have two hyper-parameters λ and γ that balance the contributions for the scale regression and skeleton localization respectively. Further information is provided in the joint detection experiments (see sec. 5.4).

Testing Phase

Out of a single input image $X = \{x_j, j = 1, \dots, |X|\}$, we will be able to obtain an edge map \hat{E} and two skeleton maps: \hat{S}_{loc} and \hat{S}_{scale} . \hat{S}_{loc} indicates which pixels are skeleton/non-skeleton in terms of a probability value, as \hat{E} does. \hat{S}_{scale} contains information about the scale of each skeleton pixel. \hat{E} is obtain as in eqn. 4.6. \hat{S}_{loc} and \hat{S}_{scale} are obtained as in eqns. 4.17 and 4.19. At the moment, we are not interested in \hat{S}_{loc} , but it will be important for the next extension of our joint approach, that is why it needs to be mentioned.

The contributions that we are going to use for both tasks (\hat{E} and \hat{S}_{loc}) are the ones provided by the fused activations. To obtain the scale map \hat{S}_{scale} we also need the information of the scale-associated side-output regression layers. Here we only want to take the previous two.

$$(\hat{E}_{fuse}, \hat{S}_{fuse},) = COMB(X, (\mathbf{W}, \mathbf{w}, \mathbf{h}, \Phi, \Psi, \mathbf{H})^*) \quad (4.23)$$

$$\hat{E} = \hat{E}_{fuse}; \quad \hat{S}_{loc} = \hat{S}_{fuse} \quad (4.24)$$

4.3.2 Implementation

The implementation for the first extension of the joint network is quite straightforward. Starting from the well-known VGG16 network, we perform the following operations:

- As in HED, we add a side-output layer that will perform the edge detection step at each stage.
- Following the implementation for LMSDS, we add two sibling side-output layers from the second stage of the network. They will perform the skeleton detection step.

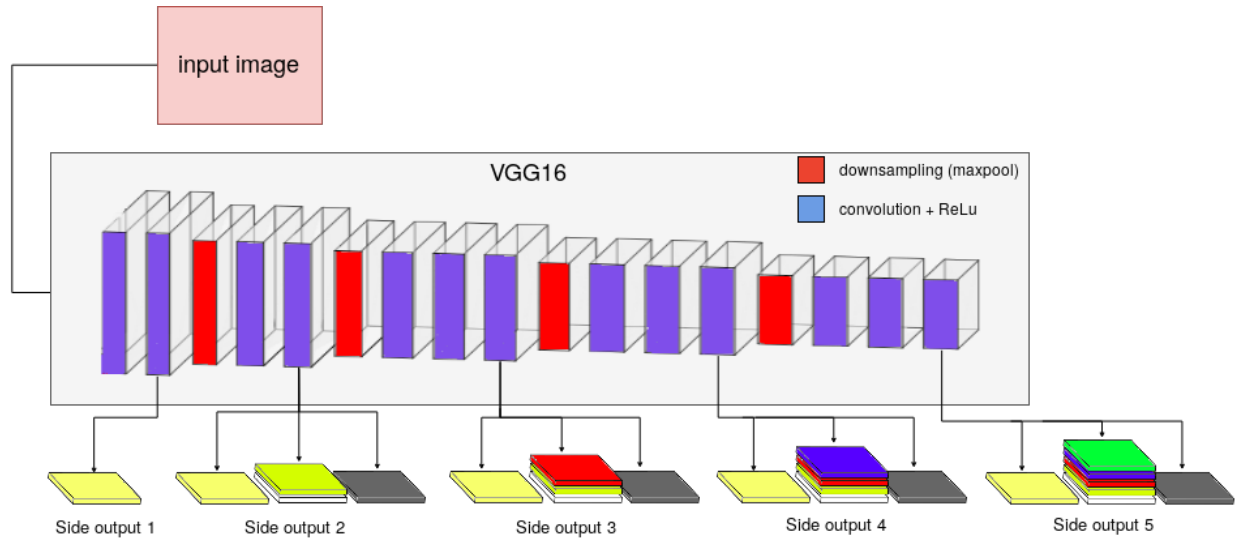


Figure 4.10: Illustration of the implementation of our joint medial axis and boundary detector. We do not include supervision as we can extrapolate it from the previous architectures (figures 4.3 and 4.8). We must not forget about the fusion layers for the edge and skeleton detection.

- In the end, we add a fusion layer for the edge detection task and another one for the skeleton detection following the same procedure defined in the previous sections.

By performing this changes we will end up with a single network that performs both skeleton and edge detection. In figure 4.10, we can observe the network that results by applying these extensions to the backbone VGG16 network..

As both detectors are contained in the same backbone network, the images that are fed will have a shared feature representation. As it has seen that there exists duality between boundaries and medial axis points, our intuitions says that this shared feature representation will help the training procedure and improve performance for both tasks.

4.4 Enforcing consistency between both tasks

The second extension to the joint medial axis and boundary detector consists of adding a module that confirms this shared feature representation mentioned above. We will do such thing by defining a consistency loss that takes into account the duality between both tasks. This last step can be achieved by using an image-to-image translation approach explained

further (see section 4.4.2). Before getting into detail about how we implement this, let's first suppose that we have two mapping functions: one lets us obtain a boundary map with information from skeletons, the other lets us get a skeleton map from a boundary map. The structure of the whole implementation can be observed in figure 4.12.

4.4.1 Formulation

Here we just define the training phase of this approach as the testing phase is a replica of the last model.

Training Phase

Let us define the loss terms that will contribute to the latest extension of the joint network. First, we group the objective functions defined in the previous section (eqn. 4.22).

$$\mathcal{L}_{comb}(\mathbf{W}, \mathbf{w}, \mathbf{h}, \Phi, \Psi, \mathbf{H}) = \mathcal{L}_{edge}(\mathbf{W}, \mathbf{w}, \mathbf{h}) + \gamma \mathcal{L}_{ske}(\mathbf{W}, \Phi, \Psi, \mathbf{H}), \quad (4.25)$$

Notice that we can obtain the edge and skeleton maps from the first extension of the joint network as seen in eqns. 4.23 and 4.24. Therefore, we build a boundary map \hat{E} and a skeleton map \hat{S}_{loc} . With this information, we could build an skeleton map (with its respective associated scales at each skeleton pixel) basing on \hat{E} symmetry. However, to build an edge map, we need information about the scales of the skeleton pixels. Here is when the previously mentioned \hat{S}_{scale} is needed as it has all the information about the scale of each skeleton pixel. In conclusion, to build a new skeleton map we just use \hat{E} and to build a new edge map we need to concatenate the information about the skeleton of the input image, $(\hat{S}_{loc}, \hat{S}_{scale})$.

Now we use the mapping functions to generate a new boundary map from the skeleton map and vice versa, \hat{E}_{new} and \hat{S}_{new} . Finally, we define a consistency loss that takes into account the new mappings in a pixel-wise fashion. We will run a non-maxima suppression algorithm (nms) to the original boundary and skeleton detections and use them as our ground truth maps, \hat{E}_{nms} and \hat{S}_{nms} . Running a nms algorithm on them makes the responses look thinner as we will see in section 5.2. The way we obtain \hat{S}_{nms} is trickier than \hat{E}_{nms} , as we are dealing with two skeleton maps: $(\hat{S}_{loc}, \hat{S}_{scale})$. To do so, we run the nms algorithm on \hat{S}_{loc} , threshold

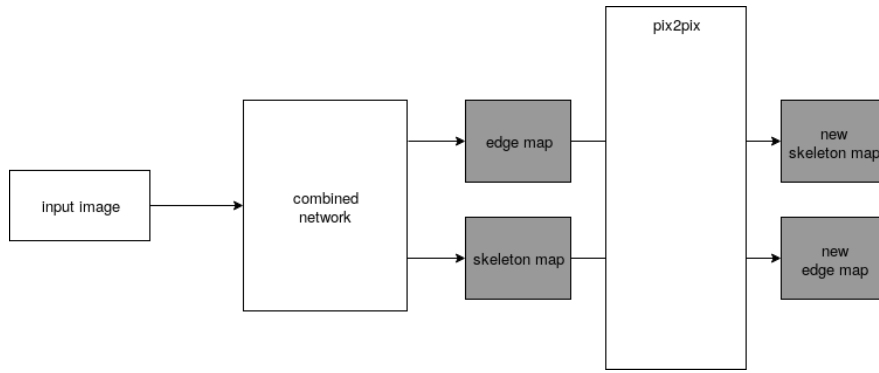


Figure 4.11: Illustration of the consistency module extension performed by image-to-image translations.

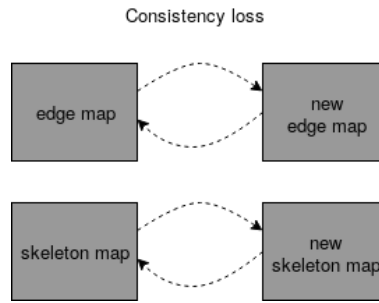


Figure 4.12: Illustration of the idea behind the consistency loss. Our intention is that the edge and skeleton predictions should be consistent when translating them into each other's domain.

it to an arbitrary value and use it as binary mask for \hat{S}_{scale}).

$$\mathcal{L}_{consistency}(\mathbf{W}, \mathbf{w}, \mathbf{h}, \Phi, \Psi, \mathbf{H}) = Dist(\hat{E}_{nms}, \hat{E}_{new}) + Dist(\hat{S}_{nms}, \hat{S}_{new}) \quad (4.26)$$

where $Dist(\cdot, \cdot)$ is an arbitrary pixel-wise loss function. For edges, we set a class balanced binary cross-entropy function (same as in eqn. 4.2) and for skeletons, a regression approach (same as in eqn. 4.11). The objective function of the joint network will be the following.

$$(\mathbf{W}, \mathbf{w}, \mathbf{h}, \Phi, \Psi, \mathbf{H})^* = \arg \min(\mathcal{L}_{comb}(\mathbf{W}, \mathbf{w}, \mathbf{h}, \Phi, \Psi, \mathbf{H}) + \mathcal{L}_{consistency}(\mathbf{W}, \mathbf{w}, \mathbf{h}, \Phi, \Psi, \mathbf{H})) \quad (4.27)$$

4.4.2 Conditional Adversarial Network

We can consider Conditional Adversarial Network (CoGAN) [17] as a network which goal is to learn a mapping function from one domain to another. The interesting thing about this framework is that we will use it to learn two different mapping functions that will let us generate skeleton maps from edge maps and vice versa. In this section we will give an overview to the formulation of the CoGAN network and we will explain the choice of the architecture to implement such mapping functions.

The image-to-image translation functions are implemented using this CoGAN framework, in which a mapping function is learned using a paired training set and an architecture based on conditional adversarial networks. Conditional adversarial networks are a variation of generative adversarial networks (GAN) [14]. It simply consists of two elements: a generator and a discriminator. The first one learns the mapping function that we are interested in whereas the latter evaluates whether the mapping has been done correctly or not. A detailed explanation of GANs is beyond the scope of this report. More information about this modern generative framework can be found here [15]. As we know, in our case we consider two domains: edges and skeletons. We will use this learning-based mapping approach and a paired edge-skeleton training set to get our image-to-image translation functions.

Training Phase

The objective of this phase is to optimize the performance for both of the elements of our framework. We will be given a training set of paired input and output images denoted as $\{(X_n, Y_n), n = 1, \dots, N\}$, respectively. We drop the subscript n for simplicity and consider one image at a time.

Recall that our framework is composed by two models. The generator learns mapping from an input image X and a random noise vector Z to its paired output image Y . In other words, the generator learns how to generate elements of the output image set from the input image set. On the other hand, the discriminator is trained to distinguish the images produced by the generator from the real ones. We denote the first element as $G(X, Z; \mathbf{W}_G)$, where \mathbf{W}_G represent the weights of the network. The latter is denoted as $D(X, Y; \mathbf{W}_D)$, where \mathbf{W}_D are also the weights of the network.

In general, the objective of the conditional GAN can be expressed as

$$\mathcal{L}_{cGAN}(\mathbf{W}_G, \mathbf{W}_D) = \mathbb{E}_{X,Y}[\log D(X, Y; \mathbf{W}_D)] + \mathbb{E}_{X,Z}[\log(1 - D(X, G(X, Z; \mathbf{W}_G); \mathbf{W}_D))], \quad (4.28)$$

where $G(X, Z; \mathbf{W}_G)$ tries to minimize this value whereas $D(X, Y; \mathbf{W}_D)$ tries to maximize it.

We also add a traditional loss that takes into account how the generated image \hat{Y} approaches to the original one Y . Therefore, the goal of the generator now is to both fool the discriminator and produce images near to the original one. The loss term that we will use is defined below and takes into account the L1 distance between Y and \hat{Y} .

$$\mathcal{L}_{L1}(\mathbf{W}_G) = \mathbb{E}_{X,Y,Z}[\|Y - G(X, Z; \mathbf{W}_G)\|_1] \quad (4.29)$$

Finally, by grouping the loss terms previously defined, the objective function can be expressed as follows.

$$(\mathbf{W}_G, \mathbf{W}_D)^* = \arg \min_{\mathbf{W}_G} \max_{\mathbf{W}_D} \mathcal{L}_{cGAN}(\mathbf{W}_G, \mathbf{W}_D) + \lambda \mathcal{L}_{L1}(\mathbf{W}_G), \quad (4.30)$$

where λ is a hyper-parameter that balances the contributions of both loss terms. As we can see, two elements appear in this objective. In our experiments, we will proceed by optimizing both elements alternatively – one gradient descent step for G , then one gradient ascent step for D .

Testing Phase

We can obtain an output image \hat{Y} by simply feeding the generator with an input image X and a random noise vector Z

$$\hat{Y} = G(X, Z; \mathbf{W}_G^*) \quad (4.31)$$

This is how the image-to image translation is performed. However, in our approach we will see that do not use the random noise vector Z to feed the generator. Consequently, our mappings become deterministic.

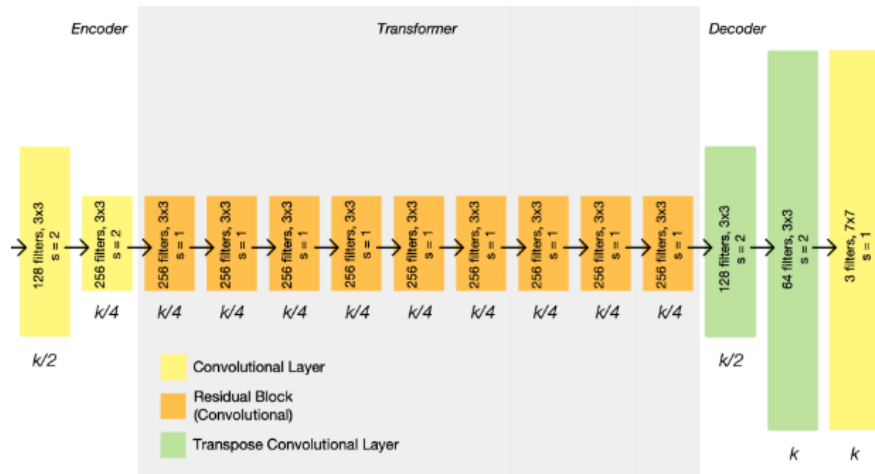


Figure 4.13: Illustration of the generator architecture, composed by three stages: encoding (convolutions), domain translation (9 residual blocks) and decoding (deconvolutions). This setting is identical for both domain translations.

Implementation

The choice of the architecture of the generator and discriminator is taken from the original CoGAN work [17], where several models are presented. Below we give an overview of the structure of both elements that compose this framework used to learn both image-to-image mapping functions.

In order to generate the domain translations, the generator consists on a network composed of three stages as shown in figure 4.13. The first stage consists on encoding the input image, which is performed by a set of convolutional layers. The next stage is the domain translation, in which the input's domain is translated into the other one (if the input is an edge map, it is translated to a skeleton map; if it is a skeleton map, it is translated to an edge map). The domain translation is performed by several residual blocks. The last stage consists on decoding the translated result into the output image. This is achieved by using transposed convolution layers.

On the other hand, the discriminator encodes the input image so as to get one value, which will represent the probability of the input image to be a real one – instead of a domain translation. The encoding is performed by several convolutional layers as shown in figure 4.14. It is important to mention that the ability of the discriminator to distinguish between good boundary and skeleton pairs lies on the receptive field size of the model. Recall that

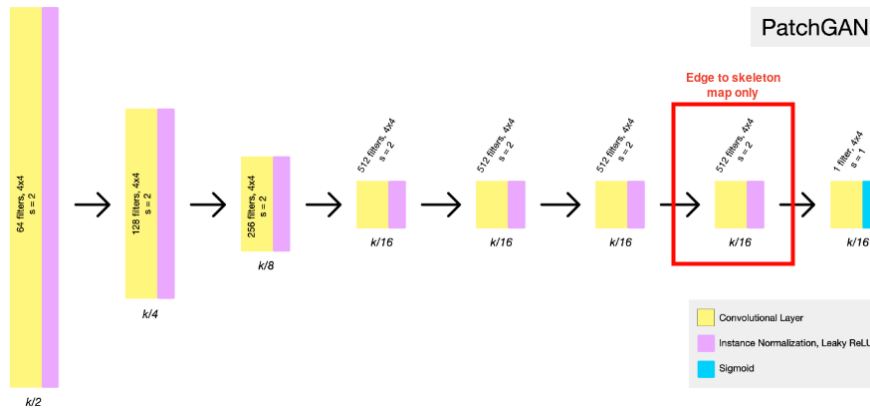


Figure 4.14: Illustration of the discriminator architecture, also called PatchGAN. It performs the discrimination on squared regions with the size of the receptive field size of the network. The red box shows that we add one extra layer for the edge to skeleton mapping because it gives better translations.

the discriminator receives information of skeletons and edges. The behaviour of this network is something similar to what we mentioned in figure 4.4: the receptive field size needs to be large enough to associate the medial axis of two boundaries of an object with its skeleton.

Preprocessing: Preparing pre-trained mapping functions for the consistency module

In this section, not only we show how the consistency module is implemented but also explain how to prepare the modules to perform the task of this last extension. When adding this module to the joint approach, it will be very difficult to obtain decent results if we train everything from scratch provided that we have three separate convolutional neural networks working jointly. Rather than this, we will start with pre-trained versions of the joint edge and skeleton detector and both image-to-image translation functions. Consequently, below is explained how to obtain the pre-trained mapping functions.

To do so, we will perform such training using the ground truth edges and skeleton maps of the same data set used in our experiments, COCO [25]. The generation of the ground truth paired images (edge and skeleton maps) will be explained in the next section). Here, we just suppose that we are given a set of paired edge and skeleton maps as shown in figure 4.15 (first and third column, respectively). The code used to develop this mapping functions consists on a modification of the code of the same authors of the CoGAN framework written

in *pytorch*: <https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/>.

The configuration of the hyper-parameters and the choice of architecture to build both mapping functions has been chosen arbitrarily. We follow [17] and use the following hyper-parameter values:

- learning rate: $2 \cdot 10^{-4}$
- momentum terms for the Adam optimizer: 0.5, 0.999
- L1 distance hyper-parameter λ : 100
- initialization value for D and G weights: normal distribution of mean 0 and standard deviation 0.02

The training scheme followed by both of the mapping functions is almost identical. The only difference is that in order to learn the mapping from skeletons to edges, we configure the optimizer to 20 steps on the generator for each step on the discriminator, otherwise we can have optimization problems due to gradient vanishing for the generator. The problem that we are dealing with is that the discriminator finds easier to perform its task than the generator. Recall that the generation task is more difficult in this case, as we need information about the skeleton scales and the skeleton/non-skeleton pixel positions.

A performance evaluation protocol of these domain translation operations is out of scope of this report. Therefore, we analyze the results qualitatively by regarding several translations performed in both directions (from edges to skeletons and vice versa), such as the ones that we show in figure 4.15. As we can see, the results look decent enough to be considered as a reasonable setting of this last extension of the joint edge and boundary detector.

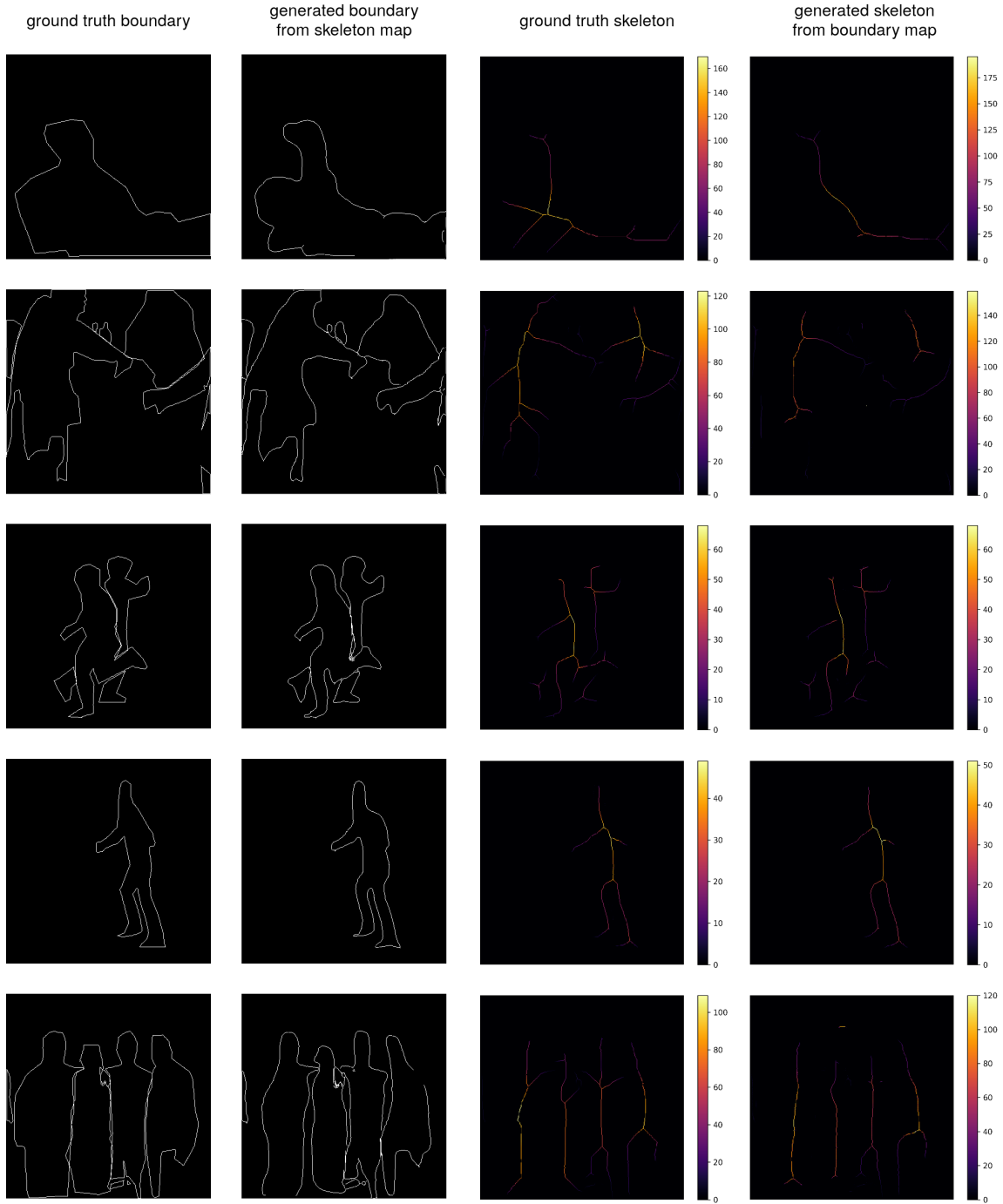


Figure 4.15: Some examples of the domain translations. Each row represents a paired edge and skeleton map. The first column shows the original boundary map; the second one is the result of mapping its respective skeleton map to the edge domain; the third one represents the original skeleton map; and the last column is the generated skeleton from its respective ground truth boundary map.

Chapter 5

Experiments

In this chapter we explain all the experiments that have been carried out in this project. Throughout its course, we have made re-implementations of both HED and LMSDS. Then we have combined the edge and skeleton detection task in a joint network and extended it by adding the consistency loss term. All the models have been implemented using the *pytorch* framework.

This section is structured as follows. First, we present the different datasets used in our experiments. Then, we explain the performance evaluation protocol followed to score the performance of the models involved in the different experiments. Later, we show the process of re-implementing HED and LSMDS and compare our results with the ones reported by their authors. Finally, we perform the experiment with our joint medial axis and boundary detector and its extension, and we compare their results with respect to the re-implementations.

5.1 Datasets

5.1.1 BSDS500

The Berkeley Segmentation dataset [4] contains a total of 500 images: 300 of them are used for training/validation purposes and the remaining 200 are used for testing. Each element of the dataset consists of a natural image and human annotations as ground truth. Each image comes with edge annotations, provided by 3–7 different annotators. An example is shown in figure 5.1. This dataset is important as it is used to evaluate the state-of-the-art algorithms

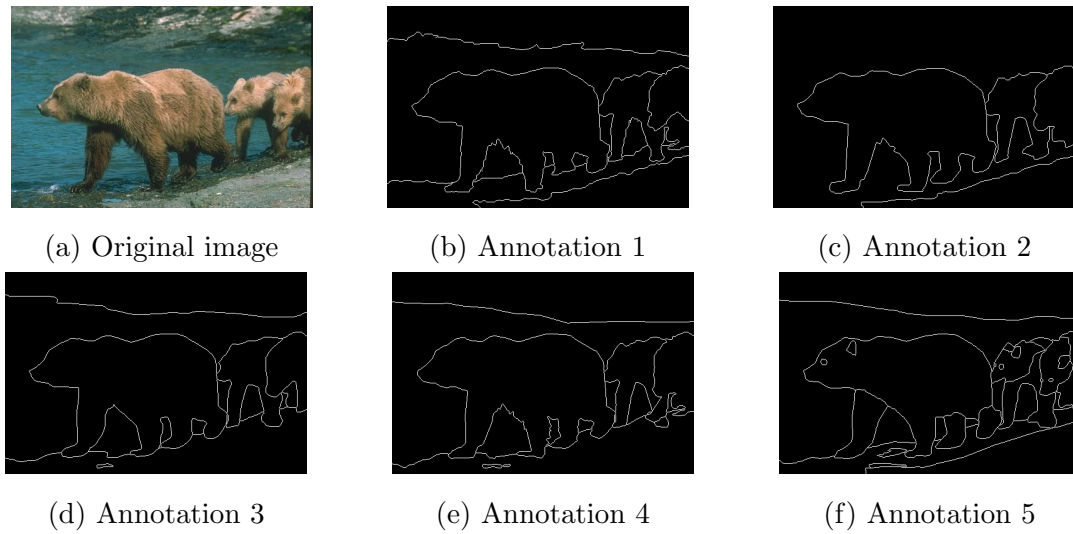


Figure 5.1: Sample from BSDS500 dataset which shows a natural image with five different human annotations for edges, denoted in white.

regarding edge detection. This dataset is used later to evaluate the HED re-implementation (section 5.3.1).

5.1.2 SK-LARGE

SK-LARGE dataset is used to evaluate the algorithms used to extract skeletons in natural images. It contains a total of 1491 natural images: 746 are used for training and the remaining 745 for testing. All the skeleton extractions have been obtained from MS COCO dataset [25]. All the images share a main property: there is a background and a foreground element in which the skeleton is annotated. Each element has the corresponding skeleton and contour map of its foreground object so that the scale map can be constructed. The images contain a variety of objects, e.g., people, planes, giraffes. In figure 5.2, we can see some samples from this dataset.

This dataset is used later to evaluate the LMSDS re-implementation (section 5.3.2). Recall that the test set should only be used to evaluate the best model out of all the possible hyper-parameter configurations. Therefore, we will create a validation set by taking 200 random images from the training set. This new set will be used to perform a fine-tuning of the hyper-parameter λ of LMSDS.



Figure 5.2: Some examples of SK-LARGE dataset. The skeleton ground truth is indicated in red.

5.1.3 COCO

Common Objects in Context (COCO) [25] is a dataset which goal is to make progress in the state-of-the-art of object recognition tasks. The dataset contains a large amount of natural scenes. Each natural scene contains several types of objects which are grouped into different categories (80 in total): *person*, *vehicle*, *animal*, etc.

The dataset includes a variety of challenges and provides the corresponding ground truth set for each of them. In our case, we focus on edge and skeleton detection tasks. However, direct ground truth maps indicating the corresponding edges or skeletons of objects is not included in this dataset. Then, we will use the information related to instance segmentation challenge, which contains ground truth maps that indicate segmentations of objects that appear in the scene. Some examples of this instance segmentation ground truth maps are shown in figure 5.3. Later, we will explain how do we generate the corresponding edge and skeleton maps given a natural scene.

The reason why COCO dataset has been chosen for our experiment is because the scenes that are contained in it display a huge variety between simple and complex scenarios. We are interested in how our joint network responds to a natural scene depending on its degree of complexity, i.e. the amount of instances that appear in it. In our experiment, we choose a version of the dataset that contains 64115 natural scenes related to *person* instances as a training set. This version also contains 2693 validation images (also related to *person* instances) that we will use as our test set. In our training/testing, we resize the input



Figure 5.3: Several samples from COCO dataset where several people appear. These people "instances" have been segmented from the original image.

natural scenes to 400x400 pixels. Moreover, we choose to target only 'person' category, due to the amount of input images that are provided and also as a starting point of evaluating our models.

5.2 Performance evaluation protocol

To score the results obtained by a given model, a performance evaluation protocol for both skeleton and edge detection is defined. It consists on building a precision-recall curve. The evaluation of both tasks is identical as we have an output detection and a ground truth label map in both cases. In figure 5.4, we show this procedure for edge detection. First, we extract all the edges of our test set. Each edge detection corresponds to a probability map which indicates the most likely pixels that correspond to edges. Then, a non-maximal suppression (nms) algorithm is applied to the edge maps to obtain thinner responses.

To construct this precision-recall curve, we calculate the precision and recall of the thinned detector responses after applying a threshold mask. Precision is defined as the number of

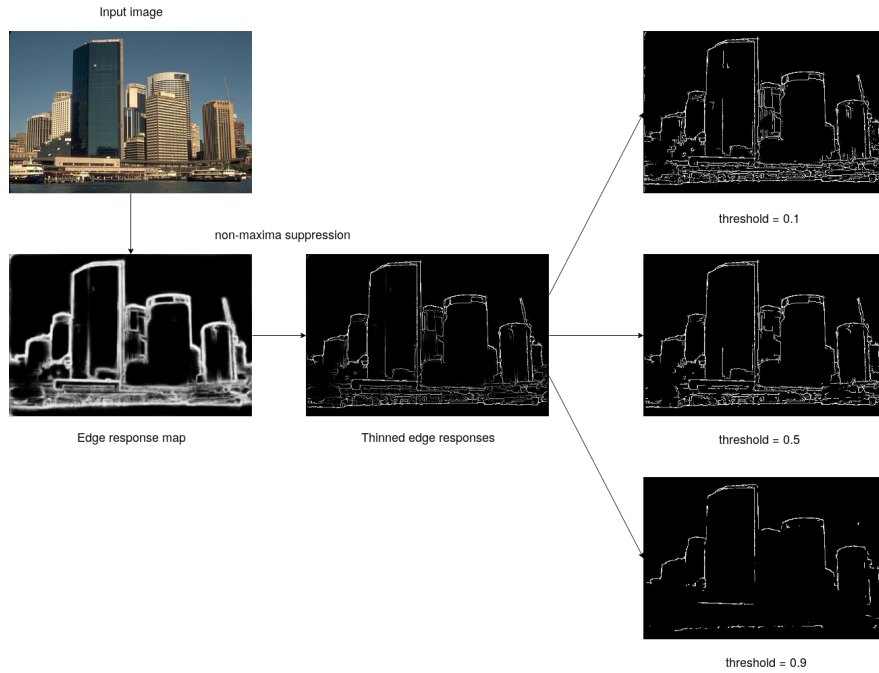


Figure 5.4: Example of the edge detection performance evaluation protocol for a single image of the test set. The valid F-score is the one which is best for a certain threshold applied to the thinned response map.

true positives over all detections made by the model.

$$P = \frac{T_p}{T_p + F_p} \quad (5.1)$$

Recall is defined as the number of true positives over all ground-truth edges.

$$R = \frac{T_p}{T_p + F_n} \quad (5.2)$$

These two quantities define the F-score, which consists of an harmonic mean between both terms and is the measure used to rank the performance of the edge detection task.

$$\text{F-score} = 2 \frac{P \cdot R}{R + R} \quad (5.3)$$

To build the precision-recall curve, we threshold the detector output at different values (see figure 5.4 for an example) and compute the precision and recall values across the entire dataset. The P-R pair that produces the highest F-score is often used to summarize the

performance of the detector.

5.3 Re-implementations

5.3.1 HED

The objective of this section is to show the re-implementation the HED algorithm. We will train the network using the *Berkeley Segmentation dataset* (BSDS500) and compare its performance with respect to the original implementation developed by its authors [45]. This experiment is useful to make sure that our re-implementation is correct and we can move on to our joint approach. All the code used to develop such algorithm can be found here: <https://github.com/charlio23/HED-pytorch>.

Training scheme

Provided that our objective is to replicate the results for HED with respect to the original implementation, the training procedure to obtain the best model is identical to the one described by its authors [45]. Therefore, there is no hyper-parameter tuning in this section.

a) *Ground truth generation*

As we have seen, there are multiple edge ground truth label maps of an image in our dataset. We follow the ground truth generation procedure as stated in the original implementation [45]: only an image pixel will be labeled as edge if it is labeled by at least three different annotators (see image 5.5); the rest will be considered as non-edge pixels.

b) *Model parameters*

Recall that we do not perform parameter tuning. Therefore, we adopt the same values for the hyper-parameters and the initialization values for the weights and biases of our network as the ones used by the authors.

The initialization value of the weights of the backbone VGG16 network of our model (denoted as \mathbf{W} in the formulation chapter 4.1.1) will be taken from the weights of a VGG16 network trained on the ImageNet challenge [9]. It has been shown that

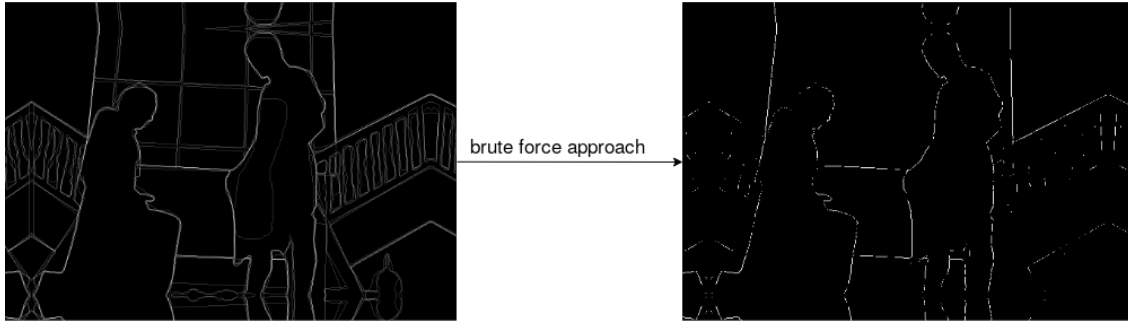


Figure 5.5: Example of the brute-force force approach taken to generate the ground truth set. The image on the left shows all the human annotations put together.

fine-tuning neural networks from models already pre-trained on the general image classification task is useful to the low-level edge prediction task, rather than starting from scratch. The configuration of the rest of the parameters of our model is listed below.

- Learning rate: 10^{-6}
- Momentum: 0.9
- Weight decay: $2 \cdot 10^{-4}$
- Initialization of the weights of the side-output layers \mathbf{w} : 0
- Initialization of the fusion layer weights \mathbf{h} : $\frac{1}{5}$
- Side-output loss weight $\{\alpha_m, m = 1, \dots, 5\}$: 1

Having set this configuration we train our model during several epochs to ensure that convergence is achieved. Furthermore, we reduce the learning rate by 10 after each epoch. We observe that performance becomes stable after 5 epochs.

c) *Data augmentation*

We adopt a pre-processed dataset published by the authors in which data augmentation has been performed. The images have been rotated to 16 different angles and cropped the largest rectangle in the resulting image. Moreover, the images have also been flipped upside down, leading to an augmentation factor of 32 times with respect to the original set.

d) *Upsampling method*

The side-out responses are upsampled using transposed convolution so as to match the

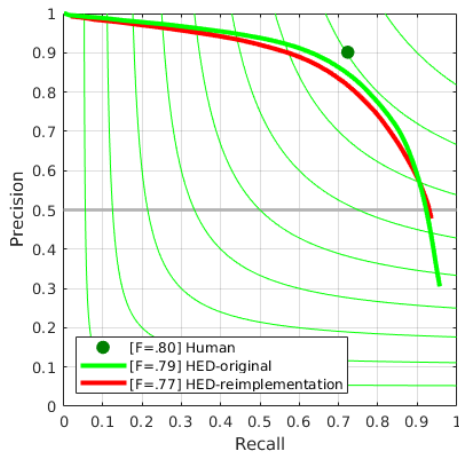


Figure 5.6: Comparison of our results on the BSDS500 dataset with respect to the original ones.

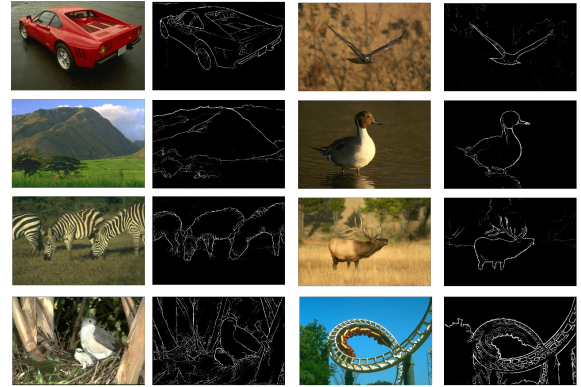


Figure 5.7: Some examples of the edge map responses obtained by our version of the HED after applying nms.

input image size. We fix their weights to perform bilinear interpolation as described in [35]. The authors report that learning the weights for the upsampling layer does not provide any improvement.

Results

After the training phase is finished, we run the trained model on the whole test dataset and perform the performance evaluation protocol described above.

In figure 5.6 we show the precision-recall curve that our re-implementation draws and compare it with respect to the one drawn by the original model. The F-score of our re-implementation 0.772 whereas the one reported by the authors is 0.790, which is slightly smaller. In conclusion, we can consider our version of the HED to be able to reproduce the results of the original one and we will use it in further experiments when comparing its performance with respect to the joint approach, which is the overall objective of this report. Moreover, in figure 5.7, we can see some examples of our version of the edge detector.

5.3.2 LMSDS

This section aims to show the performance of our re-implementation of Deep Skeleton algorithm. To do so, we use the SK-LARGE dataset [36], created by the same authors of this algorithm. In fact, we will perform two experiments.

1. In the first one, we will train the LMSDS model choosing different values for the hyperparameter λ (recall the notation used in sec. 4.2.1) to see how the trade-off between skeleton localization and scale regression affects the performance. This experiment is important because we could not find any information on the response of the model using different values for λ .
2. Eventually, we will choose a value for λ which achieves the best performance and run an experiment where the results of our best model of the re-implementation will be compared to the original one.

This experiment is necessary so as to confirm that we are able to build a skeleton detector which approaches to the original LMSDS. We will use this algorithm in the next section to compare it with our joint medial axis and boundary detector. All the code used to perform this experiment and replicate the LMSDS algorithm can be found here: <https://github.com/charlio23/DeepSkeleton-pytorch>

Training scheme

The training scheme followed by both of the experiments included in this section is almost identical. Therefore, we will only define the training scheme once and remark the differences between both of them in case they exist. As in the previous section, there is no hyperparameter tuning (except for λ) because we take the values reported by the authors as well.

a) *Ground truth generation*

The procedure to generate ground truth maps is the following. In the dataset that we are using, each image is associated with a boundary map which corresponds to the foreground object and a skeleton that indicates the medial axis points of the closure defined by the boundary. We can obtain a ground truth scale map of an object by calculating the distance in pixels from each skeleton point to its nearest edge pixel. By performing this operation across the entire training set, we can easily build our scale

maps so that they can be quantized afterwards (as defined in the formulation chapter in sec. 4.2.1).

b) *Model parameters*

As it has been mentioned previously, there is no hyper-parameter tuning except for the value of λ . The configuration of the parameters and the initialization values of the weights of our network for the training phase is the following.

The initialization of the weights of the VGG16 backbone network of our skeleton detector (denoted as \mathbf{W}) is the same that we adopted in the previous experiment. We start from a network trained on the imageNet challenge [9]. The configuration of the rest of the hyper-parameters is the following.

- Learning rate: 10^{-7}
- Momentum: 0.9
- Weight decay: $2 \cdot 10^{-4}$
- Initialization of the weights of the side-output layers Φ and Ψ : 0
- Initialization of the fusion layer weights $\mathbf{H} = \{\mathbf{h}_k, k = 0, \dots, 4\}$: $\frac{1}{n}$, where n is the length of \mathbf{h}_k .

We train our model during several epochs to guarantee the convergence of the objective function and we reduce the learning rate by 10 after each epoch as before.

We decide to run a first experiment to observe how the objective function behaves throughout the training phase. As we can see in figure 5.8, its value becomes stable from the tenth epoch.

c) *Data augmentation*

Data augmentation is performed using some code provided by the same authors. The images of the training set are rotated to four different angles (0° , 90° , 180° , 270°), flipped with three different configurations (up-down, left-right, no flip) and resized to three different scales (0.8, 1, 1.2). Consequently, we will augment the training data by a factor of 36. In other words, in the first experiment we will have 19656 training images and in the second experiment 26856.

d) *Upsampling method*

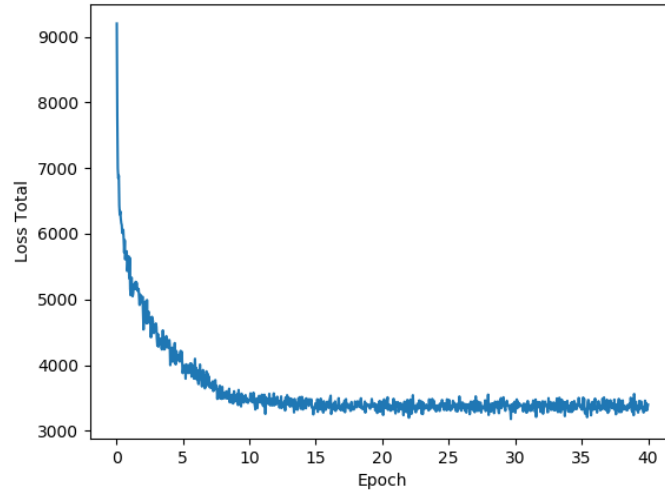


Figure 5.8: Example of the value of the objective function with respect to the epochs run in the training phase.

The upsampling weights are configured in the same way that we did with the edge detector: we fix their value so that they perform bilinear interpolation upon the side-output responses at each stage.

Results

We follow the same evaluation protocol that was described in section 5.2. Apart from generating a skeleton map of the foreground object, the LMSDS algorithm can also reconstruct the object segmentation because it preforms scale regression.

With the objective of a better evaluation of our re-implementation we will also generate segmentations out of each input image because it is the easiest way to make sure that our algorithm knows to perform scale regression from the skeleton extraction. The object segmentation of an input image can be reconstructed as follows: considering the output scale map $\hat{S} = \{\hat{s}_j, j = 1, \dots, |X|\}$ (described in the formulation, sec. 4.2.1), a segmentation map can be built by joining disks (D_j) with diameter \hat{s}_j and centered at the pixel x_j , $\bigcup_{j=1}^{|X|} D_j$. The evaluation protocol that is followed in this case consists of addressing the consistency of the output generated with respect to the ground truth object segmentation. We adopt a F-score measurement approach. First, we will take the thinned skeleton extraction (i.e. after applying nms) thresholded to the value that gives the best F-score for that instance. Then

using the thresholded skeleton map, we will build the segmentation using the information of the scales that correspond to the skeleton pixels. Finally, we will evaluate the precision and recall of the segmentation and generate a F-score value. This procedure will only generate a single score value, contrary to the precision-recall curve generated by the skeleton localization evaluation protocol.

a) *Hyper-parameter λ*

This experiment consists on training the model choosing different values for λ so as to see how the trade-off between both contributions affects performance. We run the algorithm following the training scheme indicated previously across several epochs. The values of λ are configured to see how the algorithm responds to different orders of magnitude for this hyper-parameter. The results are shown in table 5.1.

The hyper-parameter λ balances the contributions between the skeleton localization and the scale regression. It is correct to assume that if we set this value too high the skeleton localization will be poorer as the objective function will focus on the scale regression. On the other hand, the skeleton detector may not be able to learn to perform a correct scale regression if this value is too low. Most importantly, we have to consider that the scale regression is important for us to build the object segmentation. However, if the quality of the skeleton localization is poor, this will result in inaccurate object segmentations as well.

The results of this experiment follow our prior intuition: if λ is too high, the performance of the skeleton localization is affected and leads not only to inaccurate skeleton maps but also worse object segmentations. Nonetheless, we can see that if this value is too low, the performance of both tasks is still maintained.

In conclusion, the values of λ that are best for both tasks are those between 0.1 and 1. For this reason, we will take $\lambda = 0.5$ in our re-implementation and in our joint network because we are interested in both extracting skeletons and being able to recover the segmentation of the image with the highest quality possible. Predicting accurate values of the scale of the skeleton is very important to build our joint skeleton and boundary detector and enforce consistency between both tasks.

b) *Re-implementation*

After the previous fine-tuning experiment, we can build our LMSDS re-implementation.

λ	skeleton localization	object segmentation
0.0001	0.586	0.607
0.0003	0.592	0.612
0.001	0.589	0.600
0.003	0.591	0.607
0.01	0.591	0.604
0.03	0.591	0.609
0.1	0.592	0.614
0.316	0.592	0.618
1	0.589	0.622
3.16	0.555	0.591
10	0.534	0.589
31.6	0.476	0.556
100	0.392	0.472
316	0.290	0.351
1000	0.242	0.382

Table 5.1: Results of the LMSDS algorithm by choosing different values of λ . The results have been evaluated using a split of the original training set.

As done in the previous re-implementation, we run the trained model on the whole test dataset and follow the performance evaluation protocol described previously on the extracted skeletons. The precision-recall curve that is generated can be observed in figure 5.9, together with the one that is described by the original LMSDS algorithm.

As we can see, the F-score achieved by our re-implementation is 0.615: 5% lower with respect to the original model (0.649). The reason why there exists such difference may be caused by the differences that exists in the frameworks used to develop both algorithms. However, we can consider that our approach is capable of extracting good skeleton maps and we will use this skeleton detector in further experiments to compare its performance with our joint boundary and skeleton detector. Some examples of the skeleton extractions performed by our LMSDS can be seen in figure 5.10.

5.4 Joint edge and skeleton extraction

This section consists of the last and most important experiment presented in this report. Its purpose is to show the response of our joint medial axis and boundary detector in nat-

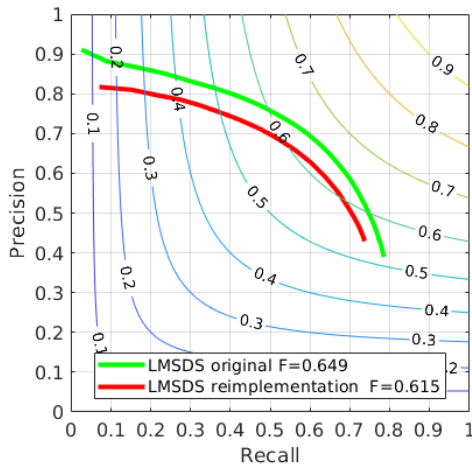


Figure 5.9: Precision-recall curve comparison between our LMSDS re-implementation and the original version, both tested on SK-LARGE dataset.



Figure 5.10: Some examples of skeleton extractions performed by our version of LMSDS after applying nms.

ural scenes and compare the performance of both tasks with respect to the other detectors. Moreover, we no longer use a dataset of simple natural images – foreground segmentation and background. Instead, the response of all the models with respect to more complex natural scenes is analyzed. The dataset that provides us the kind of scenarios that we are interested in is COCO dataset [25]. The reason why we want to use complex natural scenes in this experiment is because extracting boundaries and medial axis points is more challenging as the scenes contain more information. We believe that a shared feature representation, which is provided by the joint architecture, will be able to benefit both tasks as it takes advantage of the duality between boundaries and skeletons.

Contrary to the previous edge detection experiment, now we are focused exclusively in the edges that correspond to the boundaries of an object rather than its internal edges. The reason why we only care about object boundaries is because its symmetry points gives information about the localization of the object skeleton. On the other hand, one can obtain information about the boundary of the object if it is given a skeleton map. This consists on the duality between boundaries and skeletons that we have mentioned several times in this report, and it is the base of our joint medial axis and boundary detector.

In order to compare the results of all the models involved in this experiment, we divide the results of the experiments in different scenarios. This scenarios are organized by means of

the complexity of the scenes that are represented in the input images. We consider that the more segmented shapes that appear in the ground truth label map of its respective natural scene, the more complexity carries with it. Therefore, we take three different groups as our evaluation scenarios: Scenes with 1 segmentation, between 2 and 5 segmentations and more than 5 segmentations. We will evaluate each separately, with the respective models that are involved in it.

The models that take part in this experiment are mentioned below. For the edge detection part, we use our HED re-implementation, the joint network presented in section 4.3 and the extension of the combined network which includes the image-to-image mapping to enforce consistency between both tasks. For the skeleton extractions, we use our LMSDS re-implementation and the two versions of the joint network mentioned previously. The code used to implement our joint approaches can be found here: <https://github.com/charlio23/combined-skeleton-edge-detection>

5.4.1 Training scheme

Here is described how the models that appear in this experiment are trained. The ground truth of the natural scenes will correspond to the annotations that are related to people. We choose this setting as a starting point on analyzing the response of our models. Future work could include the response to a training scheme which includes different categories. However, this new possible scenarios are beyond the scope of this project.

Ground truth generation

As a reminder, each element of the dataset consists of a natural scene which contains several ground truth annotations. These annotations consist of segmentations of elements that correspond to different categories.

The procedure to extract the ground truth boundary and skeleton scale maps is the following. Given an element of the dataset, we use *cocoapi* mentioned above to extract the different annotations that corresponds to the "person" category. For each annotation, we run a deterministic algorithm [13] to extract the boundary and skeleton map of the corresponding segmentation. Then, we calculate the scale of each skeleton pixel as we did previously. Finally, we add all the information generated from all the annotations within the natural

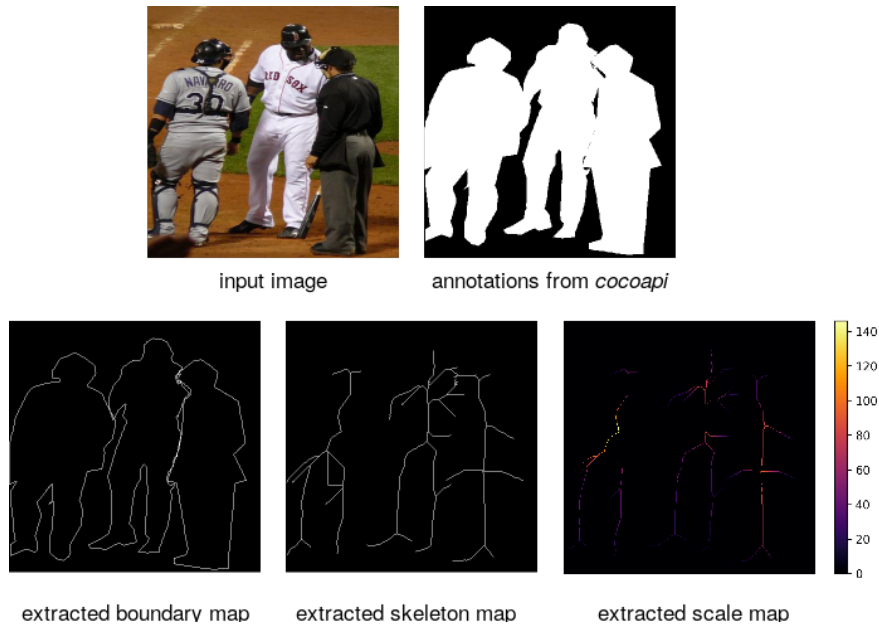


Figure 5.11: Illustration of the ground truth generation procedure referred to COCO dataset. Given a natural scene, we take its annotations from COCO to obtain object segmentations which are used to generate boundaries and skeletons in a deterministic fashion.

scene and we obtain our ground label maps. An example is shown in figure 5.11 As we are performing two tasks simultaneously, for each element of the dataset we need a boundary and a skeleton scale map.

Model parameters

Although there are several models involved in this procedure, we only state the configuration of the hyper-parameters and the value of the initialization weights of the network for the new models that participate in this experiment: the joint network and the extension with consistency between both tasks. The parameter configuration of the rest of the models – our re-implementations, are identical to the configuration stated in the previous sections. The difference is that we are using a new dataset with more complex scenes.

Due to time constraint reasons, we have not been able to optimize the hyper-parameter configuration to obtain the best possible models. However, we take an arbitrary configuration that we consider appropriate and we leave the optimization procedure as future work for this project. First we describe the initialization values for the joint network and then, the ones

for the extension that enforces consistency.

1. Joint network

As always, the initialization of the weights of the VGG16 backbone network of our skeleton detector (denoted as \mathbf{W}) is the same that we adopted in the previous experiment. We start from a network trained on the imageNet challenge [9]. The configuration of the rest of the hyper-parameters is the following.

- Learning rate: 10^{-7}
- Momentum: 0.9
- Weight decay: $2 \cdot 10^{-4}$
- Initialization of the weights of the side-output layers Φ and Ψ : 0
- Initialization of the skeleton fusion layer weights $\mathbf{H} = \{\mathbf{h}_k, k = 0, \dots, 4\}$: $\frac{1}{n}$, where n is the length of \mathbf{h}_k .
- Initialization of the edge fusion layer weights: $\frac{1}{5}$
- Skeleton regression hyper-parameter λ : 0.5
- Skeleton contribution hyper-parameter γ : 1

As done in the previous experiments, we train our model during several epochs to guarantee the convergence of the objective function and we reduce the learning rate by 10 after each epoch.

2. Joint network + consistency

The configuration adopted for this network is simple. The hyper-parameter values – of both the optimizer and the loss terms, are the same as above. The initialization values of all the components is described as follows. Recall that this extension consists of the joint network plus two image-to-image functions. The initialization values of the combined network are the ones obtained by training the joint network as defined above. The initialization of the image-to-image mapping functions is taken from the one described in section 4.4.2.

Upsampling method

As always, the weights of the upsampling layers are fixed so as to perform bilinear interpolation.

5.4.2 Results

After following the training scheme defined previously, we obtain four different models that will be tested and evaluated. As mentioned previously, we evaluate each task separately as follows.

- **Boundary detection:** The performance evaluation protocol for this task is identical to the one defined in section 5.2 at the HED re-implementation, where we draw the precision-recall curve of the result with respect to the ground truth boundary map. For this task we will generate boundary maps using these models:

1. HED
2. Joint medial axis and boundary detector
3. Joint network + pix2pix module

- **Skeleton detection:** The performance evaluation protocol in this case is identical to the one described for the skeleton extraction task for the LMSDS re-implementation. We will draw a precision-recall curve with the results provided by these models, with respect to the ground truth skeleton maps.

1. LMSDS
2. Joint medial axis and boundary detector
3. Joint network + pix2pix module

Moreover, we will check if we can recover the image segmentations using the scale regressors.

Before going into detail analyzing each scenario separately, we run the performance evaluation protocol defined for each task on the entire test set to compare the overall performance between the different models. The results are shown in figures 5.12 and 5.13.

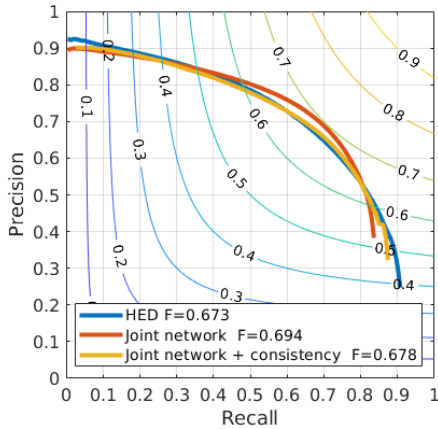


Figure 5.12: Results of the edge detection task tested on the entire COCO test set that has been defined.

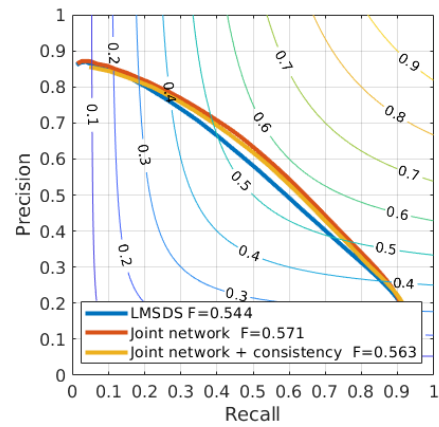


Figure 5.13: Results of the skeleton detection task tested on the entire COCO test set that has been defined.

As we can see, the overall results for both edge and skeleton detection meet our expectations. The precision-recall curves show that the shared feature representation is able to boost performance with respect to the detector that performs a single task. However, we can also observe that the joint model with the pix2pix mapping functions that enforce consistency is not able to outperform the results of the joint network.

1 segmentation

• Boundary detection:

The results for the edge detection task in this scenario can be observed in figure 5.14. As we can see, the performance of the three models is worse with respect to the entire dataset. This may be caused because in this scenario we find that 6% of the natural scenes show segmentations larger than 196 pixels wide. Recall that this value is the receptive field size of the network and we do not expect the network to perform well in this particular cases.

Nonetheless, the performance of the joint network and the one with the consistency module perform better than the HED re-implementation. Moreover, if we can see that the gap between the precision-recall curves of the joint network and the re-implementation is wider with respect to the evaluation with whole dataset.

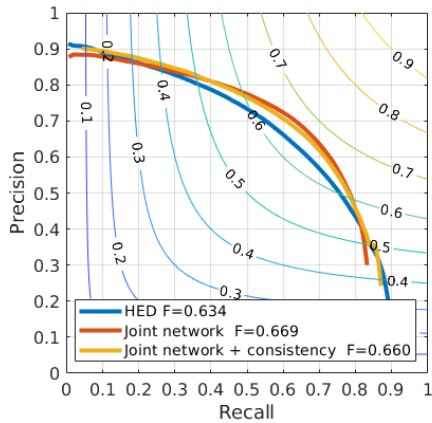


Figure 5.14: Precision-recall curves of the models defined for the edge detection task. The results correspond to the first scenario.

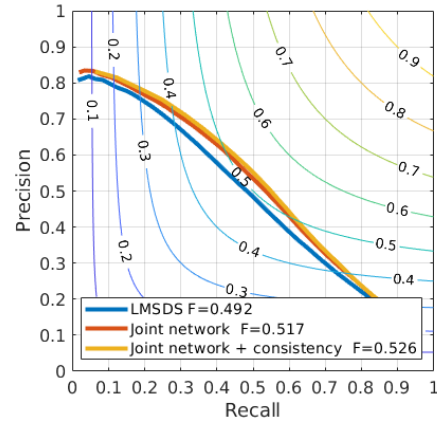


Figure 5.15: Precision-recall curves of the models defined for the skeleton detection task. The results correspond to the first scenario.

In figure 5.16, we can see some examples for this scenario.

- **Skeleton detection:**

The results for this task in this scenario are shown in figure 5.15. We can see that the shape of the precision-recall curves are similar with respect to the ones that are obtained when testing the entire dataset. Here, the joint network and the consistency extension outperform the LMSDS re-implementation as well. Contrary to the evaluation with the whole dataset, in this case we observe that the consistency module boosts the performance of the joint approach.

As found in the edge detection task, the overall performance is dropped severely, due to the scores obtained when evaluating the network with images that contain segmentations with scales higher than 196 pixels. Some examples of this task can be observed in figure 5.17

2 - 5 segmentations

- **Boundary detection:**

We can find the results for this task in this scenario in figure 5.18. As we can see, the performance of the three models is higher with respect to the results found when

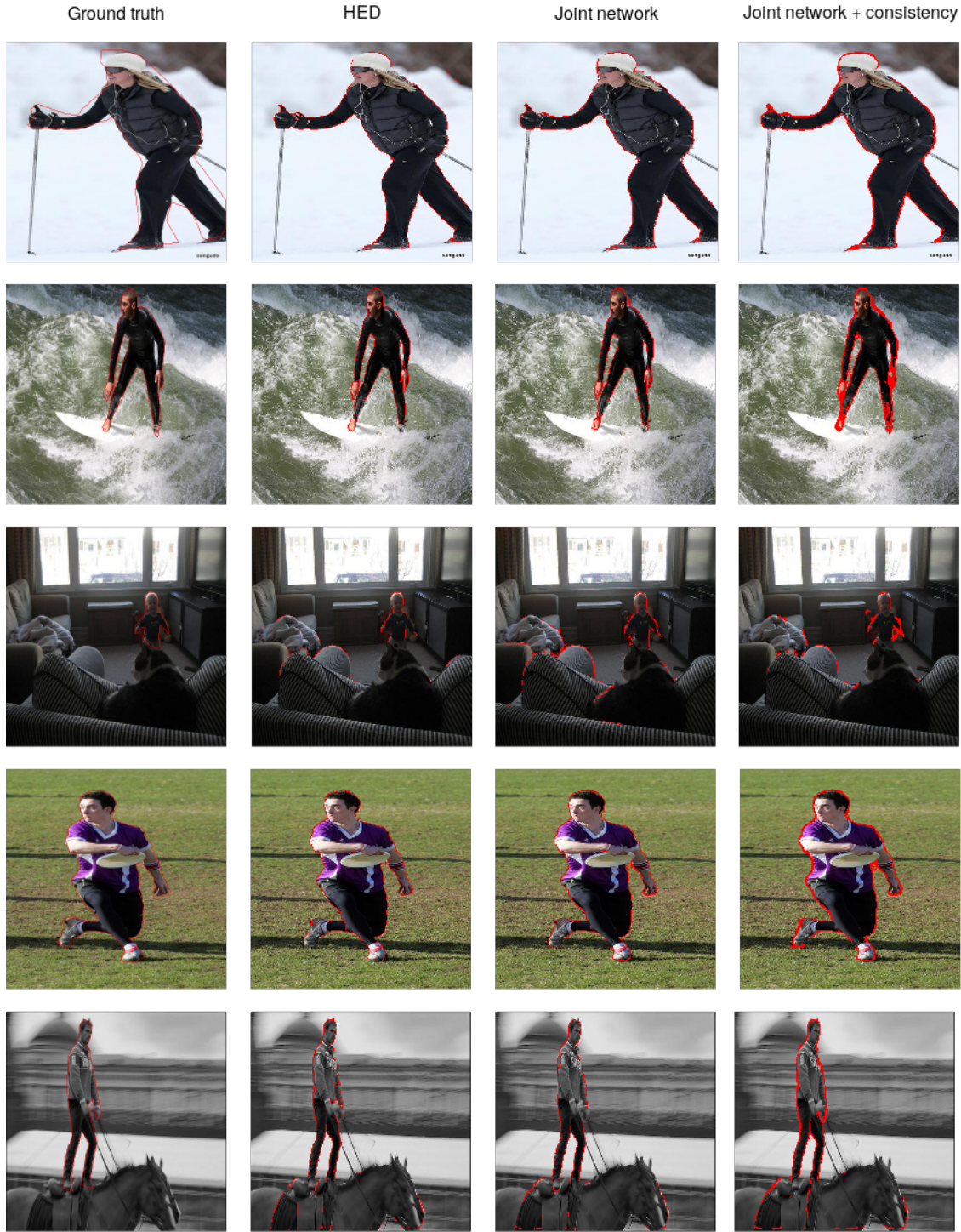


Figure 5.16: Samples of the edge detectors in the 1-segmentation scenario. We can see that when the scene shows more elements, the detectors fire incorrect edges. Moreover, the detectors are able to capture the boundary of the object without firing internal edges. Notice also that the responses of the consistency module look thicker than the first ones.

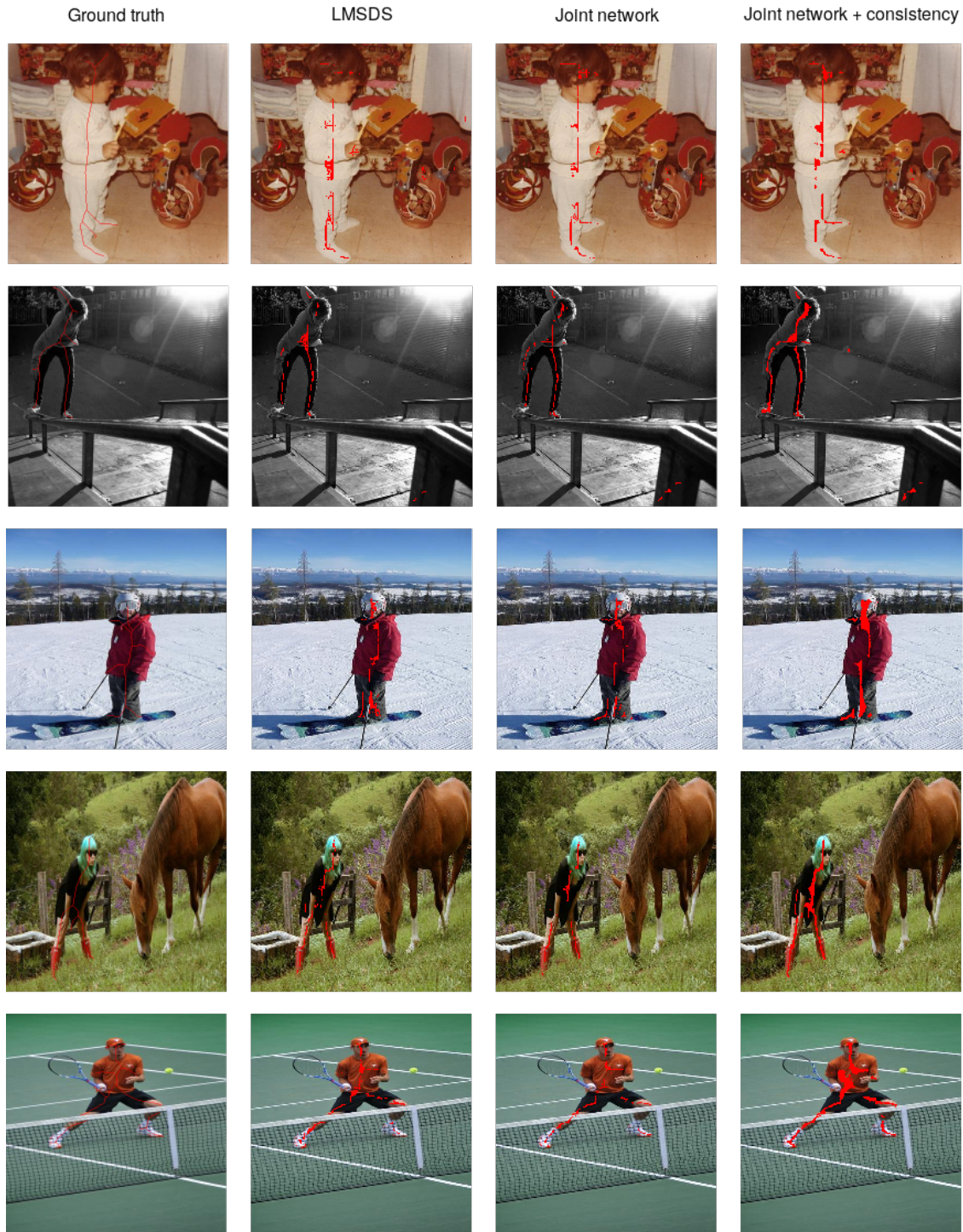


Figure 5.17: Samples of the skeleton detectors in the 1-segmentation scenario. All of them are able to extract object skeletons. Notice that in general, our joint detector provides more precision than the other ones.

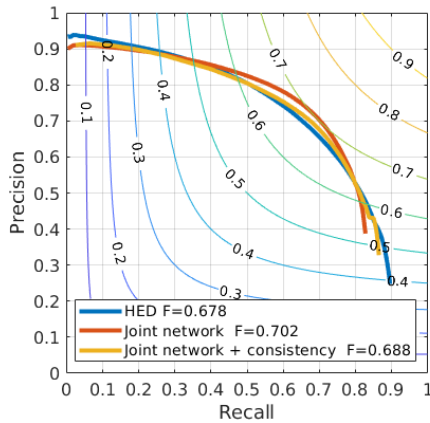


Figure 5.18: Results of the edge detection task evaluated on natural scenes that contain between 2 and 5 segmentations.

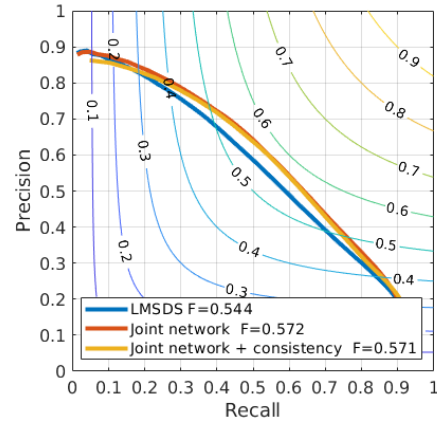


Figure 5.19: Results of the skeleton detection task evaluated on natural scenes that contain between 2 and 5 segmentations.

evaluating the whole dataset. The joint network approach provides better results as well. However, the consistency extension is not able to give better results than the previous one. Some examples of the response of the networks considering the boundary detection task are shown below (figure 5.20)

- **Skeleton detection:**

The results for the edge detection task for this scenario are shown in figure 5.19. As we can see, the results achieved by the joint model approach and its extension (consistency module) are similar. Both of them outperform the LMSDS re-implementation. As seen in the edge detection task, the overall results are better compared with the ones obtained when evaluating the whole test set. Some examples of this task in this scenario can be seen in figure 5.21

More than 5 segmentations

- **Boundary detection:**

Figure 5.22 shows the results of this task evaluated in the last and more complex scenario. Unfortunately, we find that the HED re-implementation provides better results than the joint approach plus the consistency module. However, the joint network with-

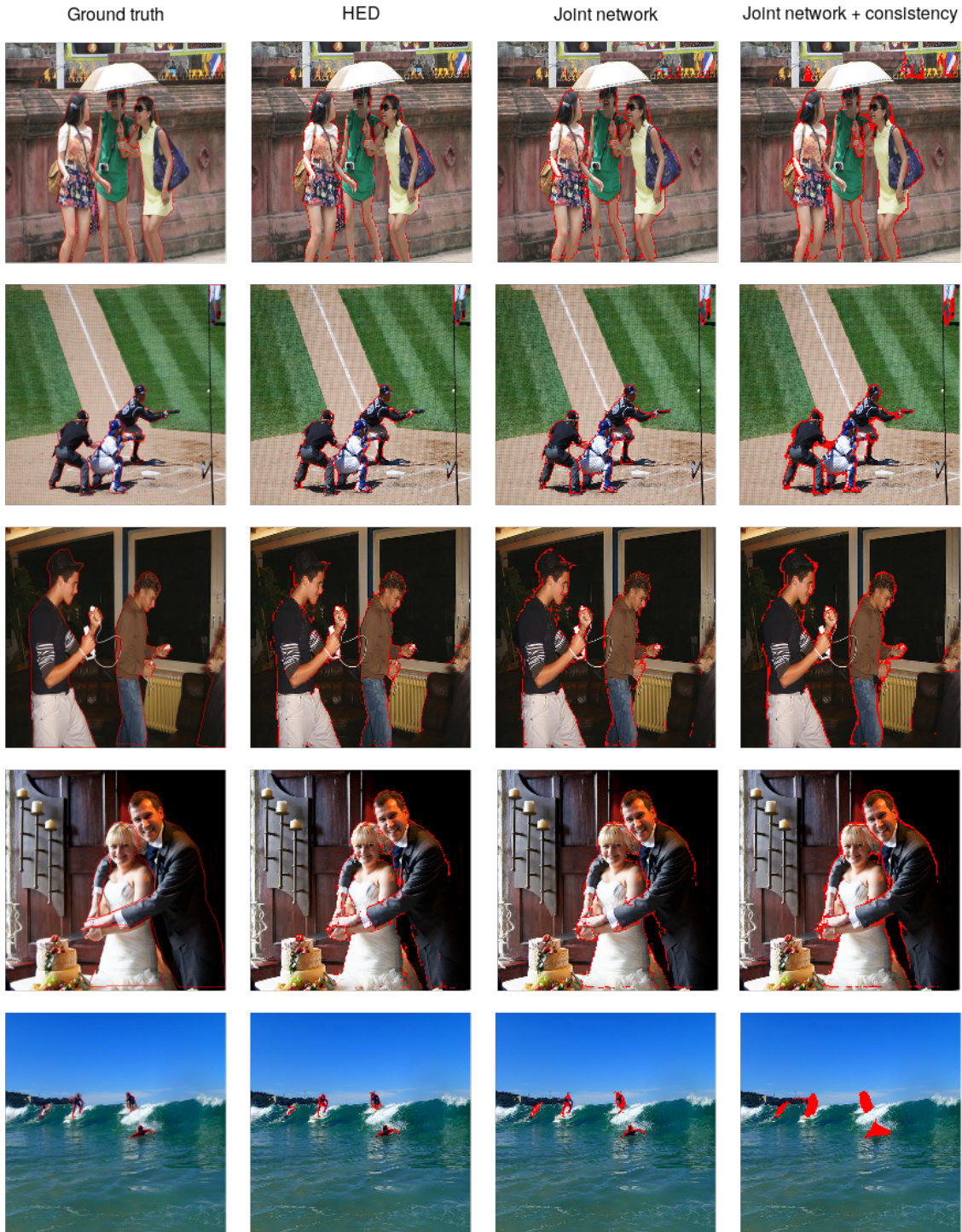


Figure 5.20: Samples of the edge detectors in the 2-to-5-segmentation scenario. These scenes look more complex than the last ones but the detectors are capable of extracting boundaries from the people that appear in the image. We can also see that our joint network with the consistency module becomes very inaccurate when the detections to fire are very small.



Figure 5.21: Samples of the skeleton detectors in the 2-to-5-segmentation scenario. Even though the task is getting more difficult, they are able to fire the main medial points. We can see that the skeletons that both joint approaches provide are more complete than the LMSDS (second row).

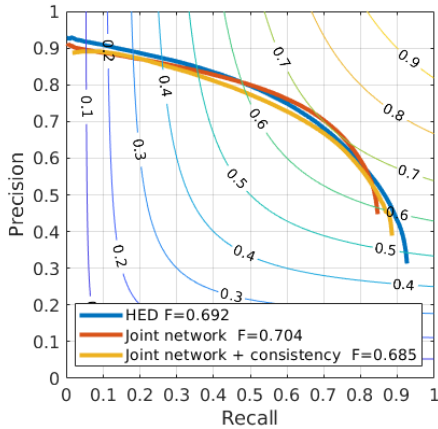


Figure 5.22: Results of the edge detection task evaluated on natural scenes that contain more than 5 segmentations.

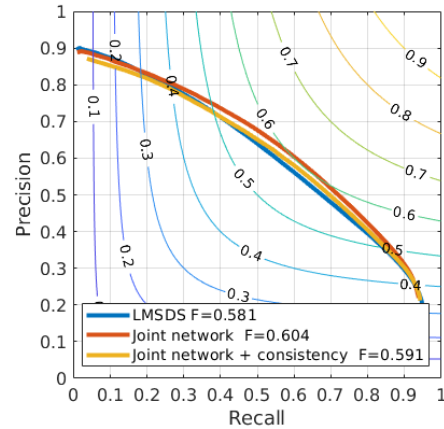


Figure 5.23: Results of the skeleton detection task evaluated on natural scenes that contain more than 5 segmentations.

out any extension is still capable of outperforming this first one. We can see some examples of the edge detection task in the last scenario in figure 5.24.

Although we find that performance is slightly better when using the joint network, we were expecting that the shared feature representation was capable of boosting the results much more. Nonetheless, this is not bad news as we have only considered one configuration of the joint approach and there is still room for parameter optimization, that could let us obtain better results.

- **Skeleton detection:**

The results of the skeleton detection task in this scenario can be seen in figure 5.23. Here, we find that the performance is similar between the three models, but our joint approach is still capable of obtaining better results. Furthermore, the overall performance is better than the one obtained when evaluating the whole test set. In figure 5.25, some examples of this task in this scenario are represented.

As we mentioned before, we were expecting better results for both of the joint network approaches due to the shared feature representation. Even so, obtaining better results just by setting an arbitrary configuration motivates us to keep exploiting the duality between edges and skeletons.



Figure 5.24: Samples of the skeleton detectors in the more-than-5-segmentation scenario.

We can see that the detectors fire all the labeled objects that appear in the scene. However, our joint network with the consistency extension provides less accuracy, which results in HED outperforming it.

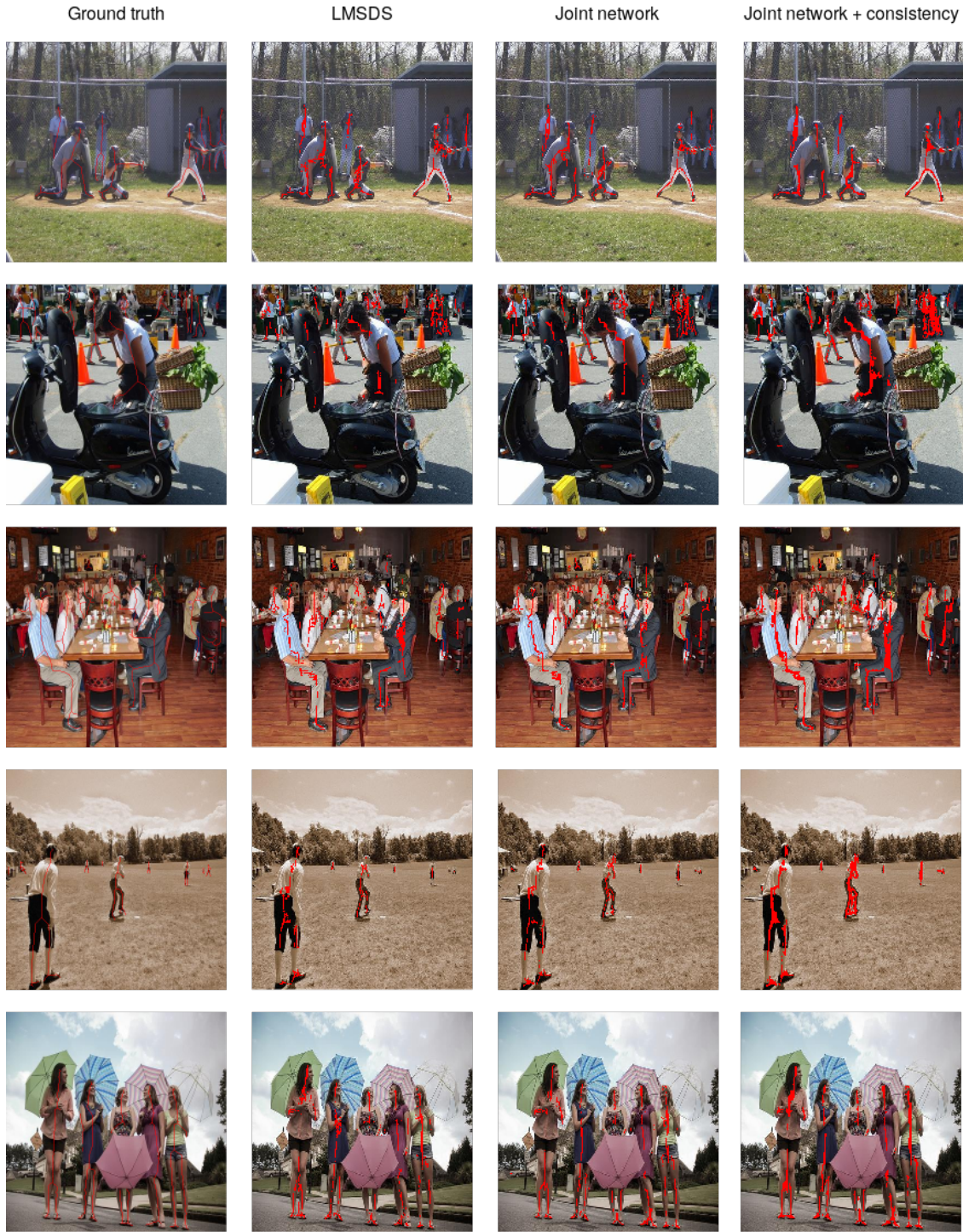


Figure 5.25: Samples of the skeleton detectors in the more-than-5-segmentation scenario. Despite the complexity that these natural scenes show, the skeleton detectors are able to fire the main symmetry points of the labels objects. In general, we see that our joint detector is more accurate.

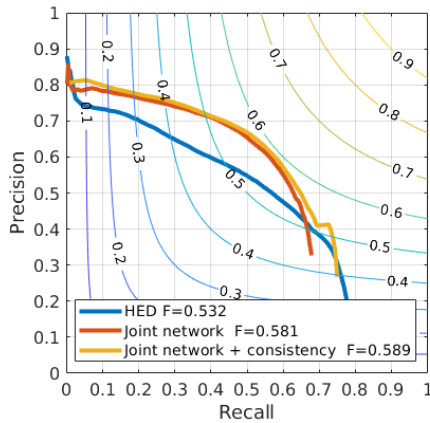


Figure 5.26: Results of the edge detection task evaluated on natural scenes where we can find segmentations wider than 196 pixels.

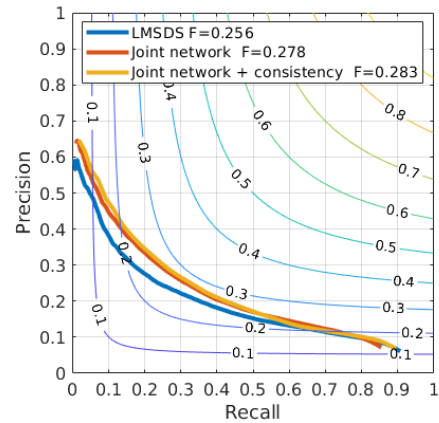


Figure 5.27: Results of the skeleton detection task evaluated on natural scenes that produce ground truth scale maps with scales larger than 196 pixels.

Scales larger than 196 pixel distance

- **Boundary detection:**

In this scenario, we expect that none of our models are able to produce reasonable results for this task. In figure 5.26 we can see that even the difficulty that this scenario presents all the models involved are capable of extracting good results. Here, our joint approach is able to outperform the HED re-implementation and the consistency module is able to slightly boost the results of the joint network. Some examples of this scenario are represented in figure 5.28.

- **Skeleton detection:**

The results of the skeleton detection of this scenario are very bad compared to the previous task (see figure 5.27). The reason why the models are not capable of extracting skeletons with enough quality with respect to the edge detection task in this scenario is the following: detecting a medial axis pixel implies that there exists symmetry at its boundaries. As we know, in this case the distance of these boundaries is larger than the receptive field size of the network at some points, making the skeleton extraction theoretically impossible.

Still, we can see that the joint approaches outperform the skeleton detector re-implementation.

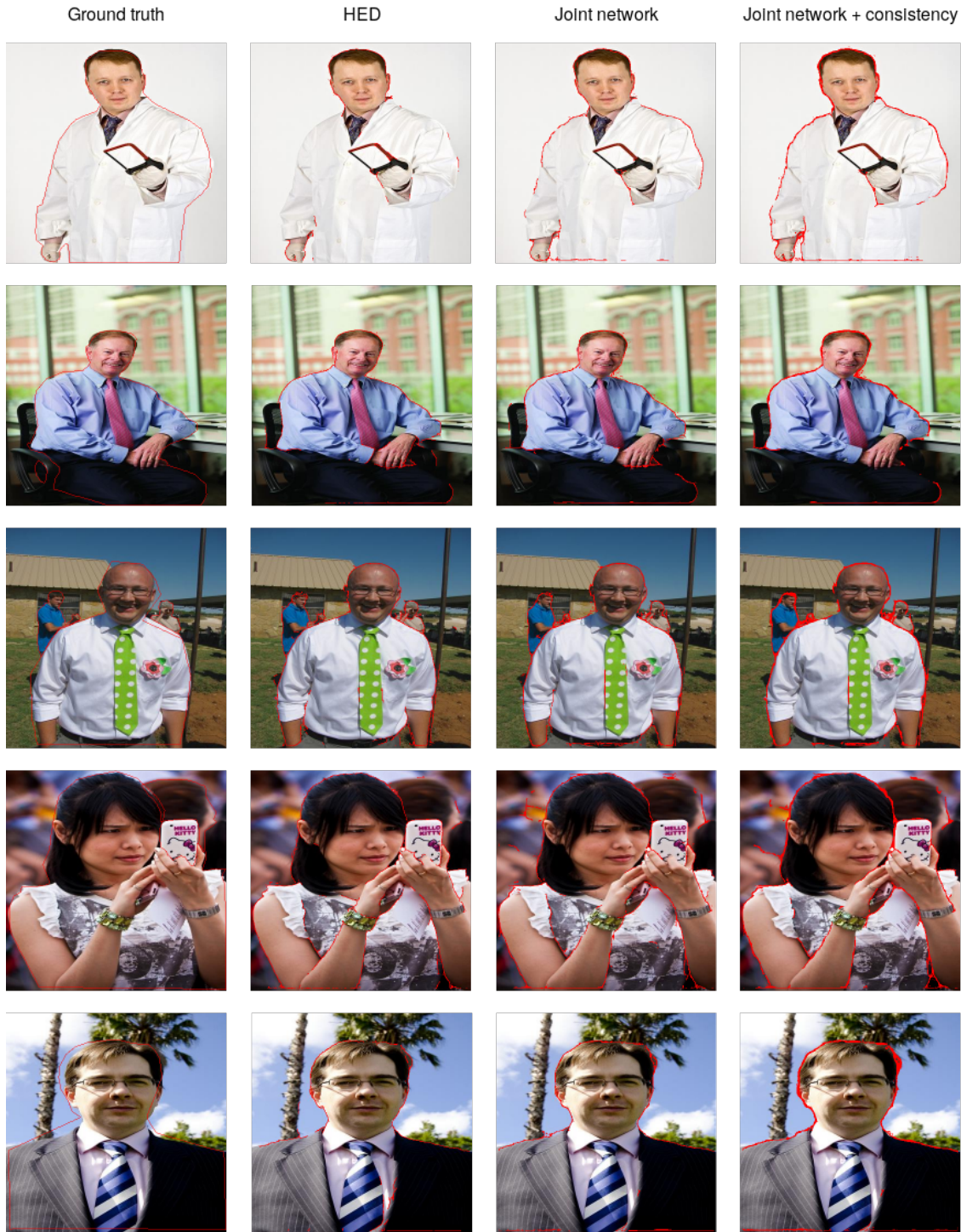


Figure 5.28: Samples of the edge detectors in natural scenes with scales higher than 196 pixels. Both joint approaches outperform HED. We can see the latter fires more internal edges. Although they are not supposed to provide decent detections, we can see that they look decent enough.

Some examples of this scenario can be observed in figure 5.29.

Image-to-Image translation

In figure 5.30 some examples of the image-to-image translation module are shown. Once we obtain the boundary and skeleton maps of an input natural scene, we forward them to our translation functions to obtain two new boundary and skeleton maps that will enforce consistency to the previous ones.

Segmentation reconstruction

As we have information about the scale of each skeleton pixel, we can reconstruct a segmentation map of the input natural scene. Some examples can be observed in figure 5.31.

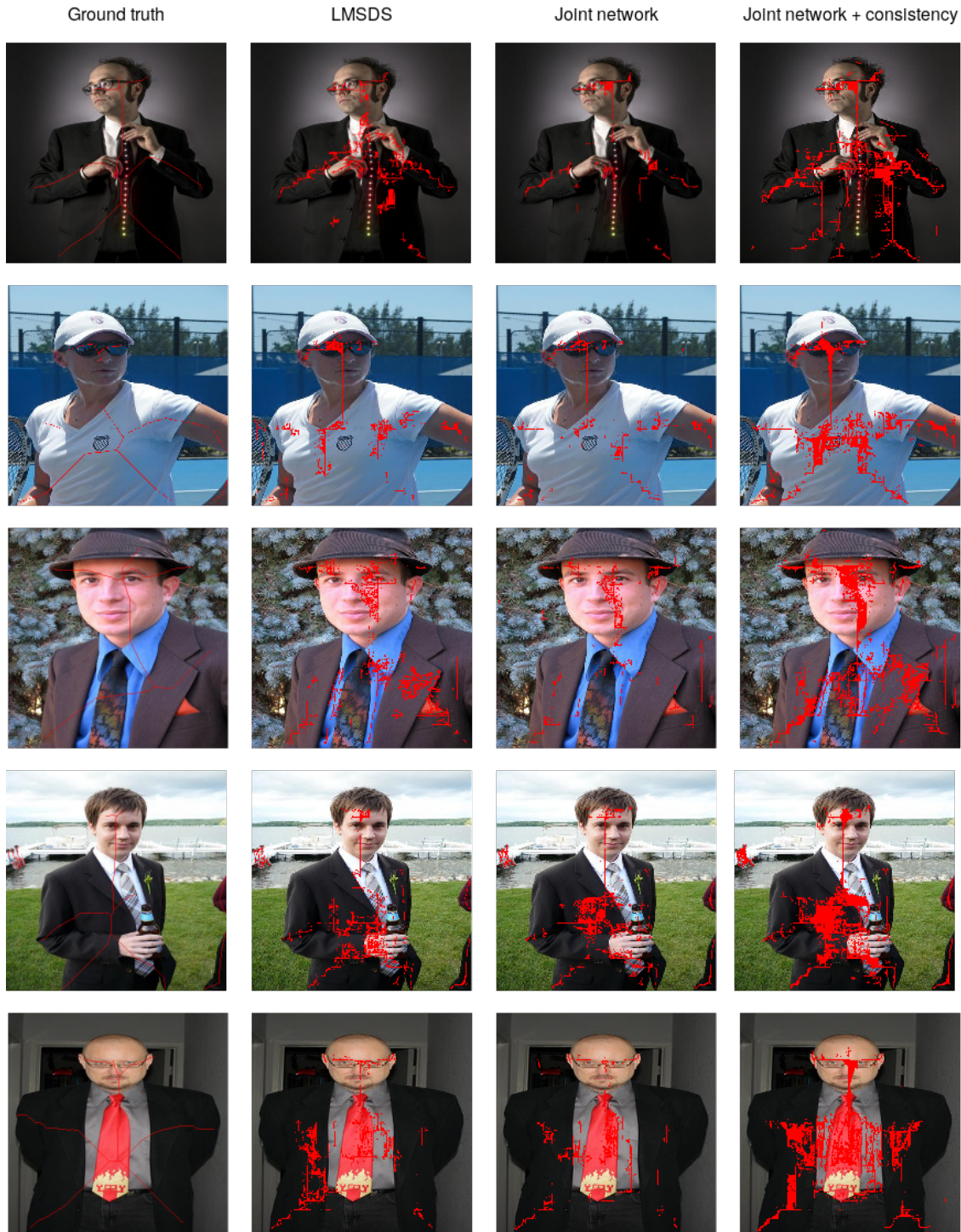


Figure 5.29: Samples of the skeleton detectors in natural scenes with scales higher than 196 pixels. We can clearly see that they are able to fire some medial axis points of the largest segmentation but in a very inaccurate manner, which results in this performance drop observed in the precision-recall curve.

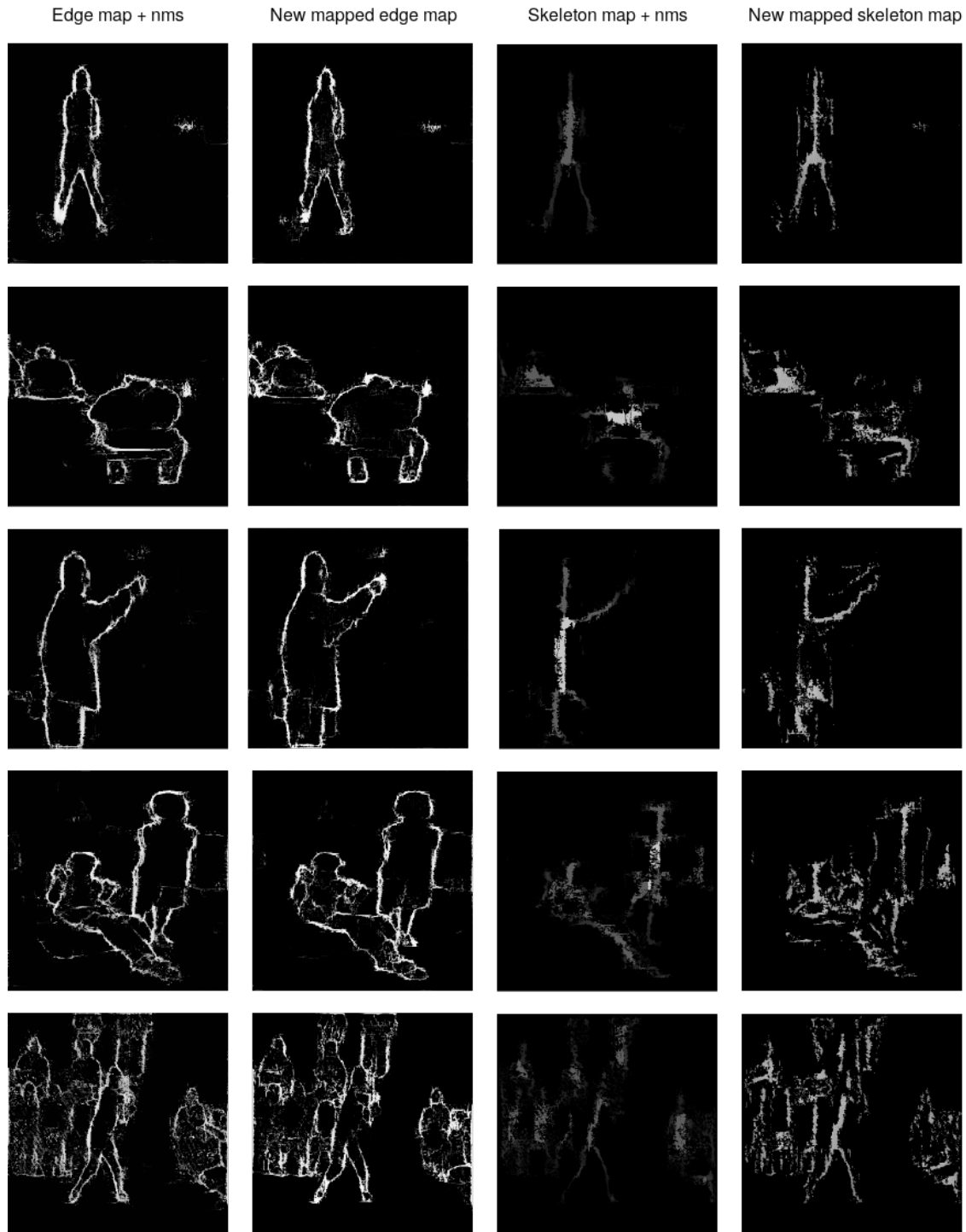


Figure 5.30: Examples showing the consistency module (one for each row). As we can see, the mapping functions let us obtain edge maps that correspond to the original ones only with information about skeleton and scales. The skeleton maps obtained from object boundaries look less accurate but look decent enough to enforce consistency. Notice that are still able to obtain decent translations when dealing with crowded scenes (last row).

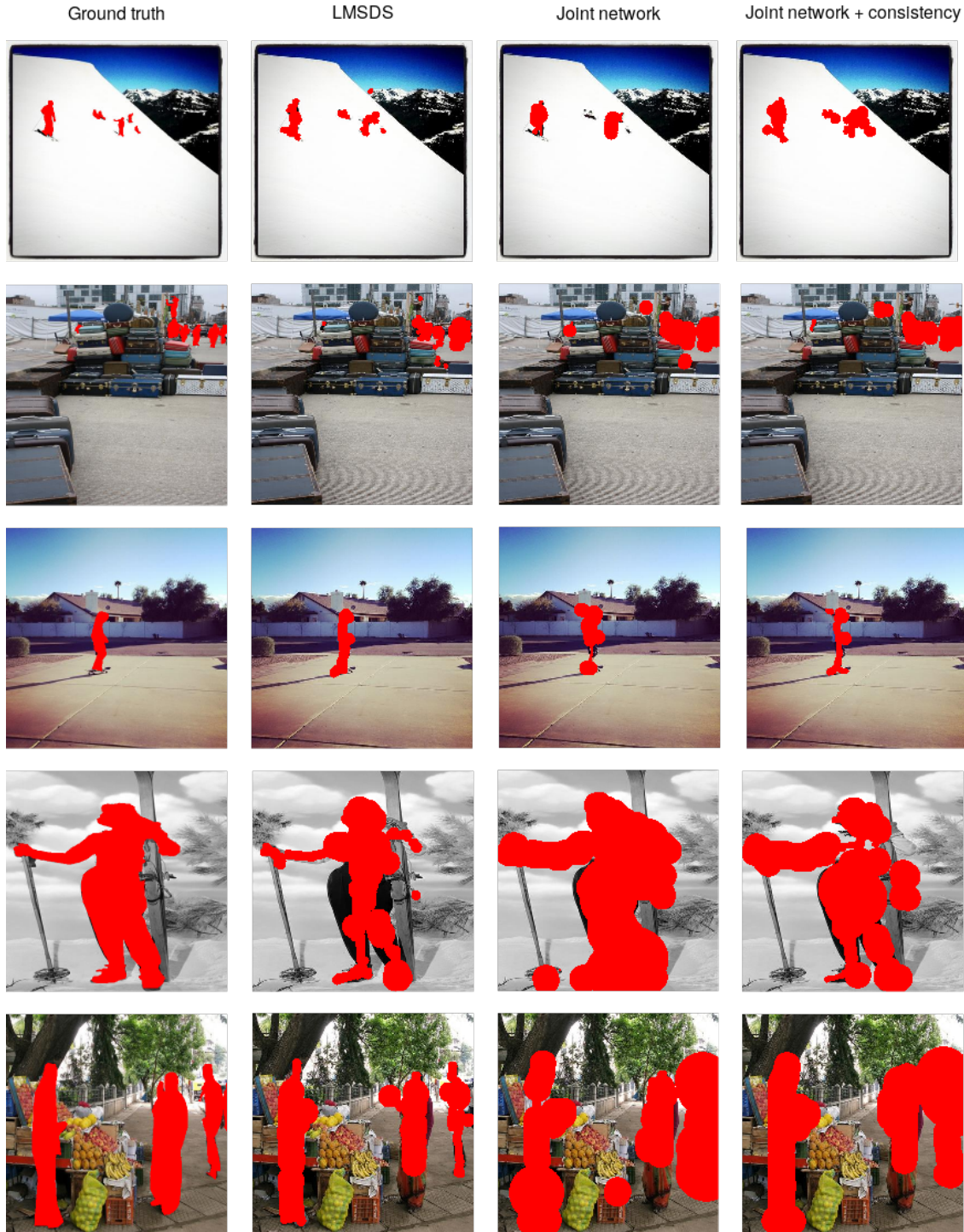


Figure 5.31: Examples of segmentation reconstruction for each object that appears in the scene. As we can see, all the models are able to segment all the objects. In general, our joint approach with the consistency module is able to provide more accurate results.

Chapter 6

Conclusions and future work

In this work we have explored the problem of jointly addressing object boundary and object skeleton extraction in real, cluttered images. We used a re-implementation of the HED [45] and LMSDS [36] frameworks, which tackle these two tasks individually, as our starting point. We then designed a framework that merges these two models, allowing for a joint feature representation and joint training for both tasks. Finally, we have complemented our joint model with an additional model that promotes consistency between the predicted boundary and skeleton maps, by means of an appropriately define consistency loss term.

Experiments on the person class from the challenging COCO dataset [25] validate the value of our joint approach. Our joint medial axis and boundary detector is capable of addressing both edge and skeleton detection simultaneously, reducing runtime and memory requirements, as it uses a single network and a shared feature representation. We also achieve better performance in both boundary and skeleton detection, with respect to HED and LMSDS, which address each one of these two tasks separately. Furthermore, our joint approach is more effective in cluttered scenes with multiple objects. A limitation of our method is that it fails to extract object skeletons when the scale of the object is larger than the receptive field of the deepest side output in our network; however, this is a limitation also shared by the vanilla LMSDS framework. Unfortunately, preliminary experiments on our consistency module did not show to further improve performance. Conducting more thorough experiments on i) plugging the two modules together; and ii) selecting optimal hyper-parameter values is part of our future work agenda.

Bibliography

- [1] Stanford cs convolutional neural networks: Linear classification.
<http://cs231n.github.io/linear-classify/>, .
- [2] Stanford cs convolutional neural networks: Architectures, convolution / pooling layers.
<http://cs231n.github.io/convolutional-networks/>, .
- [3] Stanford cs optimization: Stochastic gradient descent.
<http://cs231n.github.io/optimization-1/>, .
- [4] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011. ISSN 0162-8828. doi: 10.1109/TPAMI.2010.161. URL [here](#).
- [5] D. Attali, J.-D. Boissonnat, and H. Edelsbrunner. Stability and computation of medial axes: a state-of-the-art report. *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*, 01 2009. doi: 10.1007/b106657_6.
- [6] G. Bertasius, J. Shi, and L. Torresani. Deepedge: A multi-scale bifurcated deep network for top-down contour detection. *CoRR*, abs/1412.1123, 2014. URL [here](#).
- [7] K. Bowyer, C. Kranenburg, and S. Dougherty. Edge detector evaluation using empirical roc curves. *Comput. Vis. Image Underst.*, 84(1):77–103, Oct. 2001. ISSN 1077-3142. doi: 10.1006/cviu.2001.0931. URL [here](#).
- [8] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986. ISSN 0162-8828. doi: 10.1109/TPAMI.1986.4767851. URL [here](#).
- [9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [10] P. Dollár and C. L. Zitnick. Fast edge detection using structured forests. *IEEE Transac-*

- tions on *Pattern Analysis and Machine Intelligence*, 37(8):1558–1570, Aug 2015. ISSN 0162-8828. doi: 10.1109/TPAMI.2014.2377715.
- [11] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.
- [12] Y. Ganin and V. S. Lempitsky. N^4 -fields: Neural network nearest neighbor fields for image transforms. *CoRR*, abs/1406.6558, 2014. URL [here](#).
- [13] J. Giesen, B. Miklos, M. Pauly, and C. Wormser. The scale axis transform. In *Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry*, SCG '09, pages 106–115, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-501-7. doi: 10.1145/1542362.1542388. URL [here](#).
- [14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL [here](#).
- [15] I. J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017. URL [here](#).
- [16] J.-J. Hwang and T.-L. Liu. Pixel-wise deep learning for contour detection. 04 2015.
- [17] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.
- [18] K. R. Jerripothula, J. Cai, J. Lu, and J. Yuan. Object co-skeletonization with co-segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3881–3889, July 2017. doi: 10.1109/CVPR.2017.413.
- [19] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, Aug 2000. ISSN 0162-8828. doi: 10.1109/34.868688.
- [20] W. Ke, J. Chen, J. Jiao, G. Zhao, and Q. Ye. SRN: side-output residual network for object symmetry detection in the wild. *CoRR*, abs/1703.02243, 2017. URL [here](#).
- [21] J. Kittler. On the accuracy of the sobel edge detector. *Image and Vision Computing*, 1

- (1):37 – 42, 1983. ISSN 0262-8856. doi: [https://doi.org/10.1016/0262-8856\(83\)90006-9](https://doi.org/10.1016/0262-8856(83)90006-9). URL [here](#).
- [22] S. Konishi, A. L. Yuille, J. M. Coughlan, and S. C. Zhu. Statistical edge detection: Learning and evaluating edge cues. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(1): 57–74, Jan. 2003. ISSN 0162-8828. doi: 10.1109/TPAMI.2003.1159946. URL [here](#).
- [23] T. S. H. Lee, S. Fidler, and S. Dickinson. Detecting curved symmetric parts using a deformable disc model. In *2013 IEEE International Conference on Computer Vision*, pages 1753–1760, Dec 2013. doi: 10.1109/ICCV.2013.220.
- [24] J. J. Lim, C. L. Zitnick, and P. Dollár. Sketch tokens: A learned mid-level representation for contour and object detection. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3158–3165, June 2013. doi: 10.1109/CVPR.2013.406.
- [25] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL [here](#).
- [26] T. Lindeberg. Edge detection and ridge detection with automatic scale selection. volume 30, pages 465–470, 07 1996. ISBN 0-8186-7259-5. doi: 10.1109/CVPR.1996.517113.
- [27] C. Liu, W. Ke, F. Qin, and Q. Ye. Linear span network for object skeleton detection. *CoRR*, abs/1807.09601, 2018. URL [here](#).
- [28] Y. Liu, M. Cheng, J. Bian, L. Zhang, P. Jiang, and Y. Cao. Semantic edge detection with diverse deep supervision. *CoRR*, abs/1804.02864, 2018. URL [here](#).
- [29] P. Majer. On the influence of scale selection on feature detection for the case of linelike structures. *International Journal of Computer Vision*, 60(3):191–202, Dec 2004. ISSN 1573-1405. doi: 10.1023/B:VISI.0000036834.42685.b6. URL [here](#).
- [30] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London Series B*, 207:187–217, 1980.
- [31] D. Martin, C. Fowlkes, D. Tal, J. Malik, et al. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *Iccv Vancouver*., 2001.
- [32] D. R. Martin, C. C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis*

- and Machine Intelligence*, 26(5):530–549, May 2004. ISSN 0162-8828. doi: 10.1109/TPAMI.2004.1273918.
- [33] X. Ren. Multi-scale improves boundary detection in natural images. In D. Forsyth, P. Torr, and A. Zisserman, editors, *Computer Vision – ECCV 2008*, pages 533–545, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-88690-7.
- [34] P. K. Saha, G. Borgefors, and G. S. di Baja. A survey on skeletonization algorithms and their applications. *Pattern Recognition Letters*, 76:3 – 12, 2016. ISSN 0167-8655. doi: <https://doi.org/10.1016/j.patrec.2015.04.006>. URL [here](#). Special Issue on Skeletonization and its Application.
- [35] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, 2017. doi: 10.1109/TPAMI.2016.2572683. URL [here](#).
- [36] W. Shen, K. Zhao, Y. Jiang, Y. Wang, X. Bai, and A. L. Yuille. Deepskeleton: Learning multi-task scale-associated deep side outputs for object skeleton extraction in natural images. *CoRR*, abs/1609.03659, 2016. URL [here](#).
- [37] W. Shen, K. Zhao, Y. Jiang, Y. Wang, Z. Zhang, and X. Bai. Object skeleton extraction in natural images by fusing scale-associated deep side outputs. *CoRR*, abs/1603.09446, 2016. URL [here](#).
- [38] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.
- [39] A. Sironi, V. Lepetit, and P. Fua. Multiscale centerline detection by learning a scale-space distance transform. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2697–2704, June 2014. doi: 10.1109/CVPR.2014.351.
- [40] S. Tsogkas and I. Kokkinos. Learning-based symmetry detection in natural images. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part VII, ECCV’12*, pages 41–54, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-33785-7. doi: 10.1007/978-3-642-33786-4_4. URL [here](#).
- [41] Tyng-Luh Liu, D. Geiger, and A. L. Yuille. Segmenting by seeking the symmetry axis. In *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat.*

- No.98EX170*), volume 2, pages 994–998 vol.2, Aug 1998. doi: 10.1109/ICPR.1998.711856.
- [42] Y. Wang, Y. Xu, S. Tsogkas, X. Bai, S. J. Dickinson, and K. Siddiqi. Deepflux for skeletons in the wild. *CoRR*, abs/1811.12608, 2018. URL [here](#).
- [43] Wei Shen, Xinggang Wang, Yan Wang, Xiang Bai, and Z. Zhang. Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3982–3991, June 2015. doi: 10.1109/CVPR.2015.7299024.
- [44] N. Widynski, A. Moevus, and M. Mignotte. Local symmetry detection in natural images using a particle filtering approach. *IEEE Transactions on Image Processing*, 23(12): 5309–5322, Dec 2014. ISSN 1057-7149. doi: 10.1109/TIP.2014.2365140.
- [45] S. Xie and Z. Tu. Holistically-nested edge detection. *CoRR*, abs/1504.06375, 2015. URL [here](#).
- [46] K. Zhao, W. Shen, S. Gao, D. Li, and M. Cheng. Hi-fi: Hierarchical feature integration for skeleton detection. *CoRR*, abs/1801.01849, 2018. URL [here](#).
- [47] D. Ziou and S. Tabbone. Edge detection techniques - an overview. *INTERNATIONAL JOURNAL OF PATTERN RECOGNITION AND IMAGE ANALYSIS*, 8: 537–559, 1998.